



NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

**ON THE NUMERICAL SOLUTION OF THE INTEGRAL
EQUATION FORMULATION FOR TRANSIENT
STRUCTURAL SYNTHESIS**

by

Keenan L. Coleman

September 2014

Thesis Advisor:
Co-Advisor:

Joshua H. Gordis
Beny Neta

Approved for public release; distribution is unlimited

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188	
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.				
1. AGENCY USE ONLY (Leave blank)		2. REPORT DATE September 2014	3. REPORT TYPE AND DATES COVERED Master's Thesis	
4. TITLE AND SUBTITLE ON THE NUMERICAL SOLUTION OF THE INTEGRAL EQUATION FORMULATION FOR TRANSIENT STRUCTURAL SYNTHESIS			5. FUNDING NUMBERS	
6. AUTHOR(S) Keenan L. Coleman				
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000			8. PERFORMING ORGANIZATION REPORT NUMBER	
9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A			10. SPONSORING/MONITORING AGENCY REPORT NUMBER	
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government. IRB Protocol number ____ N/A ____.				
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited			12b. DISTRIBUTION CODE	
13. ABSTRACT Structural synthesis is the analysis of the dynamic response of a system when either subsystems are combined (substructure coupling) or modifications are made to substructures (structural modification). The integral equation formulation for structural synthesis is a method that requires only the baseline transient response, the baseline modal parameters, and the impedance of the structural modification. The integral formulation results in a Volterra integral equation of the second-kind. An adaptive time-marching scheme is utilized to solve the integral equation formulation for structural synthesis. When structural modifications of large magnitude are made, the solution to the integral equation can become unstable. To overcome this conditional stability, the derivative of the synthesis equation is examined and demonstrated to be stable regardless of the magnitude of the structural modification.				
14. SUBJECT TERMS Structural Synthesis, Adaptive Solution, Volterra Integral Equation, Trapezoidal Rule			15. NUMBER OF PAGES 109	
			16. PRICE CODE	
17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UU	

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited

**ON THE NUMERICAL SOLUTION OF THE INTEGRAL EQUATION
FORMULATION FOR TRANSIENT STRUCTURAL SYNTHESIS**

Keenan L. Coleman
Lieutenant, United States Navy
B.S., University of Arizona, 2007

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN MECHANICAL ENGINEERING

from the

**NAVAL POSTGRADUATE SCHOOL
September 2014**

Author: Keenan L. Coleman

Approved by: Joshua H. Gordis
Thesis Advisor

Beny Neta
Co-Advisor

Garth Hobson
Chair, Department of Mechanical and Aerospace
Engineering

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

Structural synthesis is the analysis of the dynamic response of a system when either subsystems are combined (substructure coupling) or modifications are made to substructures (structural modification). The integral equation formulation for structural synthesis is a method that requires only the baseline transient response, the baseline modal parameters, and the impedance of the structural modification. The integral formulation results in a Volterra integral equation of the second-kind. An adaptive time-marching scheme is utilized to solve the integral equation formulation for structural synthesis. When structural modifications of large magnitude are made, the solution to the integral equation can become unstable. To overcome this conditional stability, the derivative of the synthesis equation is examined and demonstrated to be stable regardless of the magnitude of the structural modification.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

I.	INTRODUCTION.....	1
II.	THEORY	5
A.	INTEGRAL EQUATIONS.....	5
B.	THE CONVOLUTION INTEGRAL (DUHAMEL'S).....	6
C.	IMPULSE RESPONSE FUNCTION	7
D.	INTEGRAL EQUATION FORMULATION FOR STRUCTURAL MODIFICATION	10
E.	STABILITY OF THE INTEGRAL EQUATION FORMULATION FOR STRUCTURAL SYNTHESIS	11
F.	DERIVATIVE OF THE SYNTHESIS EQUATION.....	13
III.	MODELS AND RESULTS	17
A.	GENERAL PROCESS	17
B.	SDOF MASS-SPRING SYSTEM WITH EXTERNALLY APPLIED STEP FUNCTION SUBJECTED TO A STIFFNESS MODIFICATION	18
C.	SDOF MASS-SPRING SYSTEM WITH EXTERNALLY APPLIED PERIODIC FUNCTION SUBJECTED TO A STIFFNESS MODIFICATION	23
D.	MDOF MASS-SPRING SYSTEM WITH EXTERNALLY APPLIED STEP FUNCTION SUBJECTED TO A STIFFNESS MODIFICATION	27
E.	MDOF CANTILEVERED ALUMINUM BEAM WITH AN EXTERNALLY APPLIED STEP EXCITATION, SUBJECTED TO A STIFFNESS MODIFICATION TO AN ARBITRARY BEAM ELEMENT	31
F.	THE DERIVATIVE OF THE TRANSIENT STRUCTURAL SYNTHESIS EQUATION FOR THE MDOF CANTILEVERED ALUMINUM BEAM WITH AN EXTERNALLY APPLIED STEP EXCITATION, SUBJECTED TO A STIFFNESS MODIFICATION TO AN ARBITRARY BEAM ELEMENT.....	35
G.	THE MATRIX FORMULATION FOR THE DERIVATIVE OF THE TRANSIENT STRUCTURAL SYNTHESIS EQUATION FOR THE MDOF CANTILEVERED ALUMINUM BEAM WITH AN EXTERNALLY APPLIED STEP EXCITATION, SUBJECTED TO A STIFFNESS MODIFICATION TO AN ARBITRARY BEAM ELEMENT	38
IV.	CONCLUSION AND RECOMMENDATIONS	43
	APPENDIX A. COARSE SOLUTION FOR THE INTEGRAL EQUATION FORMULATION FOR STRUCTURAL SYNTHESIS	45

APPENDIX B. THE DERIVATIVE OF THE INTEGRAL EQUATION FORMULATION FOR STRUCTURAL SYNTHESIS	47
APPENDIX C. COARSE SOLUTION TO THE DERIVATIVE OF THE INTEGRAL EQUATION FORMULATION FOR STRUCTURAL SYNTHESIS.....	49
APPENDIX D. THE MATRIX FORMULATION TO THE DERIVATIVE OF THE TRANSIENT STRUCTURAL SYNTHESIS EQUATION	51
APPENDIX E. MATLAB CODE FOR A SDOF MASS-SPRING SYSTEM WITH EXTERNALLY APPLIED STEP FUNCTION SUBJECTED TO A STIFFNESS MODIFICATION (SUPPORTING FUNCTIONS INCLUDED)...	55
APPENDIX F. MATLAB CODE FOR A SDOF MASS-SPRING SYSTEM WITH EXTERNALLY APPLIED PERIODIC FUNCTION SUBJECTED TO A STIFFNESS MODIFICATION (SUPPORTING FUNCTIONS INCLUDED)...	63
APPENDIX G. MATLAB CODE FOR THE FINITE ELEMENT MODEL OF THE ALUMINUM CANTILEVERED BEAM	71
APPENDIX H. MATLAB CODE FOR A NON-ADAPTIVE, NON-DERIVATIVE APPROXIMATION OF THE ALUMINUM CANTILEVERED BEAM (SUPPORTING FUNCTIONS INCLUDED).....	75
APPENDIX I. MATLAB CODE FOR A NON-ADAPTIVE, DERIVATIVE APPROXIMATION OF THE ALUMINUM CANTILEVERED BEAM (SUPPORTING FUNCTIONS INCLUDED).....	79
APPENDIX J. MATLAB CODE FOR THE MATRIX FORMULATION TO THE DERIVATIVE OF THE INTEGRAL FORMULATION FOR TRANSIENT STRUCTURAL ANALYSIS (SUPPORTING FUNCTIONS INCLUDED)	85
LIST OF REFERENCES.....	89

LIST OF FIGURES

Figure 1.	SDOF mass-spring with stiffness modification	18
Figure 2.	Modified response for a SDOF mass-spring with externally applied step excitation subjected to 10% stiffness modification.	19
Figure 3.	Step size for a SDOF mass-spring with externally applied step excitation subjected to 10% stiffness modification.	20
Figure 4.	Modified response for a SDOF mass-spring with externally applied step excitation subjected to 50% stiffness modification.	20
Figure 5.	Step size for a SDOF mass-spring with externally applied sinusoidal excitation subjected to 50% stiffness modification.	21
Figure 6.	Modified response for a SDOF mass-spring with externally applied step excitation subjected to 100% stiffness modification.	21
Figure 7.	Step size for a SDOF mass-spring with externally applied sinusoidal excitation subjected to 50% stiffness modification.	22
Figure 8.	Modified response for a SDOF mass-spring with externally applied sinusoidal excitation subjected to 10% stiffness modification.	24
Figure 9.	Step size for a SDOF mass-spring with externally applied step excitation subjected to 10% stiffness modification.	25
Figure 10.	Modified response for a SDOF mass-spring with externally applied sinusoidal excitation subjected to 50% stiffness modification.	25
Figure 11.	Step size for a SDOF mass-spring with externally applied step excitation subjected to 50% stiffness modification.	26
Figure 12.	Modified response for a SDOF mass-spring with externally applied sinusoidal excitation subjected to 100% stiffness modification.	26
Figure 13.	Step size for a SDOF mass-spring with externally applied step excitation subjected to 100% stiffness modification.	27
Figure 14.	Two DOF mass-spring system subjected to stiffness modification.	27
Figure 15.	Modified response for a two DOF mass-spring with externally applied step excitation subjected to 10% stiffness modification.	28
Figure 16.	Step size for a two DOF mass-spring with externally applied step excitation subjected to 10% stiffness modification.	29
Figure 17.	Modified response for a two DOF mass-spring with externally applied step excitation subjected to 10% stiffness modification.	29
Figure 18.	Step size for a two DOF mass-spring with externally applied step excitation subjected to 50% stiffness modification.	30
Figure 19.	Modified response for a two DOF mass-spring with externally applied step excitation subjected to 100% stiffness modification.	30
Figure 20.	Step size for a two DOF mass-spring with externally applied step excitation subjected to 50% stiffness modification.	31
Figure 21.	Aluminum cantilevered beam of five elements with an elemental stiffness change to the third element subjected to a step external excitation.	32

Figure 22.	Instability of the aluminum cantilever beam with a stiffness modification of 14% and a time-step of 0.01 second.	33
Figure 23.	Instability of the aluminum cantilever beam with a stiffness modification of 50% and a time-step of 0.01 second.	33
Figure 24.	Stability of the aluminum cantilever beam with a stiffness modification of 14% and a time-step of 0.0001 second.	34
Figure 25.	Stability of the aluminum cantilever beam with a stiffness modification of 50% and a time-step of 0.00001 second.	35
Figure 26.	MDOF derivative approximation with a 100% stiffness modification and a 0.01 second time-step exhibiting stability.	36
Figure 27.	MDOF derivative approximation 14% elemental stiffness modification and a time-step of 0.01 second.	36
Figure 28.	DOF derivative approximation 50% elemental stiffness modification and a time-step of 0.01 second.	37
Figure 29.	DOF derivative approximation 14% elemental stiffness modification and a time-step of 0.00001 second.	37
Figure 30.	DOF derivative approximation 50% elemental stiffness modification and a time-step of 0.00001 second.	38
Figure 31.	Matrix formulation of the derivative of the synthesis equation for the Aluminum cantilevered beam with a 100% stiffness modification and time-step of .00001 second.	41

LIST OF TABLES

Table 1.	Response to stiffness modifications for SDOF mass-spring system subjected to externally applied step excitation.	19
Table 2.	Response to stiffness modifications for SDOF mass-spring system subjected to externally applied sinusoidal excitation.	24
Table 3.	Response to stiffness modifications for a two DOF mass-spring system subjected to externally applied step excitation.	28
Table 4.	Time required to complete each part of the matrix formulation of the derivative to the integral formulation for structural synthesis.	40

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF ACRONYMS AND ABBREVIATIONS

DOF	degree-of-freedom
EOM	equation-of-motion
IRF	impulse response function
MDOF	multiple degree-of-freedom
SDOF	single degree-of-freedom
VIDE	Volterra integro-differential equation
VIE	Volterra integral equation

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

I would like to thank God for helping me find peace during a time that is full of turbulence. I would like to thank my wife, Mariesa, for her sacrifices that made everything to this point possible. I would like to thank my daughter, Annabelle, and my son, Gabriel, for keeping life in perspective. I also would like to thank Dr. Gordis and Dr. Neta for their guidance and patience during this process. Finally, I would like to thank Dr. Richard Feynman, whose marriage of genius and common sense is a huge inspiration to me.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

The theory of integral equations is a rich and complex theory that touches every branch of science and engineering. Despite its prevalence, few attempt to master the theory of integral equations. The history of integral equations dates back to the early nineteenth century when the profound mathematical insights of Newton and Leibniz were being used in conjunction with rapid technological advances to describe physical phenomena that had defied scholars for millennia. While most texts offer a brief history of integral equations in the introduction, Lonseth [1] provides a paper that is rich with history and mathematical rigor.

A fundamental problem in mechanical engineering is the efficient prediction for transient responses of components and structures, especially after modifications have been made. The design/analysis cycle often requires repeated analyses, as the designer looks for an optimal design. Structural synthesis is the analysis of the dynamic response of a system when either subsystems are combined (substructure coupling), or modifications are made to subsystems (structural modification) [2]. Structural modification can include changes in properties such as mass, stiffness or damping. Substructure coupling can occur for many reasons. It can be a natural product of the system itself as a result of factors such as material differences or complex geometries. Substructure coupling is frequently used when multiple internal or external organizations individually contribute subsystems to a governing system design. The goal when conducting this analysis is to maximize computational efficiency, a key requirement to remain competitive on a global scale.

There are many methods for structural synthesis and the method this paper focuses on is the integral equation formulation for transient structural synthesis as formulated by Gordis in [2]. This formulation is beneficial because it eliminates the process of reconstructing and resolving large systems of equations after a modification to the structure has been made. Also, the response to the modified system is based on a pre-modified, or “baseline”

system response usually calculated from a pre-existing model. This bypasses the process of reassembling and solving for the solution to the modified system. Furthermore, synthesis can reduce the scope of the analysis from the total structure, to the portions of the structure in which the modification has been made. Responses can be calculated from any portion of the structure that has not been modified but is of interest.

The integral equation formulation to structural synthesis is based on the convolution integral and results in a Volterra integral equation (VIE) of the second kind. Integral equations usually cannot be solved analytically and one must resort to numerical schemes. Since integral equations can be re-written in the form of differential equations, it implies that same numerical methods used to solve differential equations can be used to solve integral equations. Accuracy varies with the various methods, but in general, the higher the order of accuracy, the more complex the method. The approach this paper uses to solve the integral formulation for transient structural synthesis is an adaptive method based on the trapezoidal rule formulated by Gordis and Neta [3].

This method computes the whole integral as the sum of integrals evaluated on sub-intervals between the limits of integration. The trapezoidal rule is applied to each of these sub-integrals and when summed together forms the coarse solution. A fine solution is created by dividing the final sub-interval into two equal intervals, raising the accuracy level of the final interval and therefore the overall solution. This is necessary only for the final interval because all other previously calculated solutions are assumed to have been calculated to within an acceptable level of accuracy. The final interval is the only interval that has the value from the coarse solution in which the accuracy is in question. If the error between the fine and course solution meets the specified tolerance, the fine solution is stored as the coarse solution, the time-step is doubled, and the process repeated for the next interval. If the error is greater than the tolerance, the solution at the current point is deleted, the time-step is halved, and the process repeated from the last point of acceptable accuracy.

The adaptive trapezoidal method is beneficial because the level of accuracy can be established and the time-step then adjusted as necessary to meet the desired level of accuracy. The only unknown that must be determined by this method is the intermediate function value in the final interval which is a straightforward calculation. Higher order methods require a larger number of intermediate values in the final interval and therefore a higher number of intermediate function values that need to be solved. The higher level of accuracy normally obtained by higher order methods is obtained by the adaptive trapezoidal rule when the time-step is sufficiently small, or when the time-step is reduced in order to meet the set error tolerance. In [4], Linz gives a detailed analysis of the trapezoidal rule and other higher order methods, along with the truncation error of each method.

Initially the goal was to verify that the adaptive trapezoidal formulation would be a robust alternative to solving synthesis problems. After successfully duplicating the results of [3], the adaptive method was successfully applied to the synthesis of a single degree-of-freedom (SDOF) mass-spring system with a stiffness modification, and a two DOF mass-spring system with a stiffness modification. Both of these problems were subjected to step and periodic forcing functions. However, when applying this method to an aluminum cantilever beam subjected to a step forcing function with an elemental stiffness change, the method exhibited conditional stability. Upon further investigation, this conditional stability is attributed to the magnitude of the stiffness modification.

Stability analysis of integral equations is complex and is not the focus of this paper. For those interested, the issue is addressed in great detail in [5], [6], and [7]. In general, integral equations with large kernels are not unconditionally stable [7]. In structures with modifications of sufficient magnitude, the problem becomes conditionally stable. Since the modification is a component of the kernel; therefore the magnitude of the kernel can become large enough to drive the problem from unconditional stability to conditional stability.

In order to overcome conditional stability, the derivative of the synthesis equation was taken, producing a Volterra integro-differential equation (VIDE) of the second kind. Stability of the original synthesis equation and the motive for taking the derivative will be discussed in greater detail in the Theory section of the paper. Though taking the derivative complicates the adaptive algorithms, taking the derivative of the synthesis equations has three benefits:

- 1) Provides unconditional stability to the synthesis equation,
- 2) Provides velocity information that was previously unavailable, and
- 3) The solution to the original synthesis equation is inherent in the algorithm when solving for the derivative.

II. THEORY

The main focus of this analysis will be single degree-of-freedom and multiple degree-of-freedom (MDOF) systems that have undergone structural modifications; specifically stiffness modifications. Also, this analysis is strictly limited to linear models. The theory of transient structural modification, and the associated governing VIE, is based on the convolution integral, or Duhamel's integral. The convolution integral is commonly used when analyzing a system response to an external excitation. For structural modifications that are of sufficient magnitude this VIE is conditionally stable relative to the magnitude of the stiffness modification, but it will be shown that converting the VIE to a Volterra integro-differential equation provides unconditional stability; allowing the use of the adaptive trapezoidal rule to solve the system.

A. INTEGRAL EQUATIONS

Prior to discussing the integral formulation of structural synthesis, it is essential to understand the basic structure of integral equations in general terms. The following discussion can be found in greater detail in any text on integral equations, but this discussion follows closely from [8].

An integral equation is expressed generally as:

$$h(x)u(x) = f(x) + \int_0^{b(x)} K(x, \xi)u(\xi)d\xi \quad (1)$$

When $b(x)$ is constant, the equation is classified as a Fredholm equation and is not the focus of this paper. When $b(x) = x$, the equation is classified as a Volterra equation. Furthermore, if $h(x) = 0$, then both Fredholm and Volterra equations are of the first kind. If $h(x) = c$, where c is a constant, then both Fredholm and Volterra equations are of the second kind. As previously mentioned, the convolution integral used to formulate structural synthesis takes the form:

$$u(x) = f(x) + \int_0^x K(x, \xi) u(\xi) d\xi \quad (2)$$

a Volterra equation of the second kind where $f(x)$ is a known function, $K(x, \xi)$ is the known kernel function, and u is the function to be solved for.

B. THE CONVOLUTION INTEGRAL (DUHAMEL'S)

The convolution integral, or Duhamel's integral expression can be used to describe the response of a system to an external excitation and is based on the principle of superposition; which is only applicable to linear systems [9]. The Duhamel's integral expression in terms of dynamic response is as follows:

$$\{x(t)\} = \{x(t)\}_h + \int_0^t [h(t-\tau)] \{f^e(\tau)\} d\tau \quad (3)$$

where $\{x(t)\}$ is a $n \times 1$ vector of the total system response (where n is the number of DOF of the system), $\{x_h(t)\}$ is a $n \times 1$ vector of the free response of the system, $[h(t-\tau)]$ is an $n \times n$ impulse response function (IRF) matrix, and $\{f^e(\tau)\}$ is a $n \times 1$ vector of external forces (external loading) applied to the system.

In terms of $\{x_h(t)\}$, the solution of the free response will vary depending on whether the system is underdamped, critically damped, or overdamped. The equation-of-motion (EOM) for a linearly viscous damped SDOF system is:

$$\ddot{x} + 2\zeta\omega_n\dot{x} + \omega_n^2x = 0 \quad (4)$$

and assuming the solution to the ordinary differential equation is an exponential, the characteristic polynomial can be solved and the eigenvalues determined from the following:

$$\lambda_{1,2} = \zeta\omega_n \pm \sqrt{\zeta^2 - 1} \quad (5)$$

It can be seen that the magnitude of the damping ratio (ζ) determines the character of the free response EOM. There are three possible cases for the damping ratio: 1) $0 < \zeta < 1$ (underdamped), 2) $\zeta = 1$ (critically damped), and 3) $\zeta > 1$ (overdamped). Substituting the eigenvalues into the exponential solution and applying initial conditions gives the solution for the three cases of damping.

$$x(t) = e^{-\zeta\omega_n t} \left(x_0 \cos(\omega_d t) + \frac{\dot{x}_0 + \zeta\omega_n x_0}{\omega_d} \sin(\omega_d t) \right) \quad (\text{underdamped}) \quad (6)$$

$$x(t) = [x_0 + (\dot{x}_0 + \omega_n x_0)t] e^{-\zeta\omega_n t} \quad (\text{critically damped}) \quad (7)$$

$$x(t) = e^{-\zeta\omega_n t} \left[x_0 \cosh(\omega^* t) + \frac{\dot{x}_0 + \zeta\omega_n x_0}{\omega^*} \sinh(\omega^* t) \right] \quad (\text{overdamped}) \quad (8)$$

In any case, following the assumptions that $\dot{x}_0 = x_0 = 0$, this leads to $\{x_h(t)\} = \{0\}$

and therefore the convolution integral becomes:

$$x(t) = \int_0^t h(t-\tau) f(\tau) d\tau \quad (9)$$

In general, structural systems are underdamped and therefore will be the focus of this paper.

C. IMPULSE RESPONSE FUNCTION

It is important to understand the impulse response function matrix because it is the kernel of the VIE formulation of the synthesis equation; and is the component of the VIE that determines stability of the system.

The EOM of a MDOF, proportionally damped system with an external excitation applied to a single DOF is as follows:

$$[M]\{\ddot{x}\} + [C]\{\dot{x}\} + [K]\{x\} = \{p(t)\} \quad (10)$$

Often this system of equations are coupled, in other words, there are not enough independent equations to solve for the entire system. A transformation is

introduced that takes the EOM from physical coordinates to modal coordinates. This transformation uncouples the equations allowing for the appropriate number of independent equations to solve for the system. The transformation that takes the EOM from physical coordinates to modal coordinates is as follows:

$$\{x\} = [\Phi]\{q\} \quad (11)$$

Substituting Equation (11) into Equation (10), and multiplying the EOM by $[\Phi]^T$, gives the following EOM in modal coordinates:

$$[\mathcal{M}]\{\ddot{q}\} + [\mathcal{C}]\{\dot{q}\} + [\mathcal{K}]\{q\} = [\Phi]^T \{p(t)\} \quad (12)$$

where, $[\mathcal{M}] = [\Phi]^T [M] [\Phi]$, $[\mathcal{C}] = [\Phi]^T [C] [\Phi]$, and $[\mathcal{K}] = [\Phi]^T [K] [\Phi]$. Assuming that the system is mass-normalized, and the external excitation acts at a single DOF, Equation (12) becomes:

$$\{\ddot{q}\} + \begin{bmatrix} \ddots & & \\ & 2\zeta\omega_n & \\ & & \ddots \end{bmatrix} \{\dot{q}\} + \begin{bmatrix} \ddots & & \\ & \omega_d^2 & \\ & & \ddots \end{bmatrix} \{q\} = [\Phi]^T \begin{Bmatrix} 0 \\ 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \\ 0 \end{Bmatrix} p(t) \quad (13)$$

where 1 is at the j^{th} DOF. Equation (13) then reduces to:

$$\{\ddot{q}\} + \begin{bmatrix} \ddots & & \\ & 2\zeta\omega_n & \\ & & \ddots \end{bmatrix} \{\dot{q}\} + \begin{bmatrix} \ddots & & \\ & \omega_d^2 & \\ & & \ddots \end{bmatrix} \{q\} = \{\phi_j\} p(t) \quad (14)$$

Since the system in modal coordinates is uncoupled, each equation can be solved independently. The j^{th} modal EOM can be written as:

$$\ddot{q}_i + 2\zeta_i \omega_{n_i} \dot{q}_i + \omega_{n_i}^2 q_i = \phi_j^{(i)} p(t) \quad (15)$$

In order to determine the i^{th} impulse response function Equation (15) needs to be re-written as:

$$\ddot{q}_i + 2\zeta_i \omega_{n_i} \dot{q}_i + \omega_{n_i}^2 q_i = \begin{cases} p(t)\phi_0^{(i)}, & 0 < t < t_d \\ 0, & t_d < t \end{cases} \begin{matrix} (a) \\ (b) \end{matrix} \quad (16)$$

where t_d is the time the impulse is applied. Recall the impulse is defined as:

$$I = \int_0^{t_d} p(t) dt \quad (17)$$

Taking the integral of Equation 16(a) from 0 to t_d :

$$\int_0^{t_d} \ddot{q}_i(t) dt + \int_0^{t_d} 2\zeta_i \omega_{n_i} \dot{q}_i(t) dt + \int_0^{t_d} \omega_{n_i}^2 q_i(t) dt = \int_0^{t_d} \phi_j^{(i)} p(t) dt \quad (18)$$

Given the initial rest conditions of $q(0) = 0$ and $\dot{q}(0) = 0$, Equation (18) becomes:

$$[\dot{q}(t_d) - \dot{q}(0)] + 2\zeta_i \omega_{n_i} [q(t_d) - q(0)] + \omega_{n_i}^2 q_{avg} t_d = I \phi_j^{(i)} \quad (19)$$

Applying the initial conditions and understanding that q_{avg} is very small while the impulse is applied, the last term can be ignored and Equation (19) becomes:

$$\dot{q}(t_d) + 2\zeta_i \omega_{n_i} q(t_d) = I \phi_j^{(i)} \quad (20)$$

In the short duration the impulse is applied, $q(0) = 0$ and $\dot{q}(0) = I \phi_j^{(i)}$ which become the initial conditions when solving for Equation 16(b). Solving Equation 16(b) yields:

$$q_i(t) = \frac{I \phi_j^{(i)}}{\omega_{d_i}} e^{-\zeta_i \omega_{n_i} t} \sin(\omega_{d_i} t) \quad (21)$$

Letting the impulse, $I = 1$, the modal IRF is:

$$\hat{h}_i(t) = \frac{\phi_j^{(i)}}{\omega_{d_i}} e^{-\zeta_i \omega_{n_i} t} \sin(\omega_{d_i} t) \quad (22)$$

To transform the modal IRF to the physical IRF, Equation (11) is substituted into Equation (21). Applying this transformation to the system of modal IRF gives the physical IRF represented in matrix form as:

$$[h(t)] = [\Phi] \begin{bmatrix} \ddots & & \\ & \hat{h}(t) & \\ & & \ddots \end{bmatrix} [\Phi]^T \quad (23)$$

From Equation (23) it follows that the IRF matrix entries take the form of:

$$h_{ij}(t-\tau) = \sum_{p=1}^n \phi_i^p \phi_j^p \frac{1}{\omega_d} e^{-\zeta \omega_n (t-\tau)} \sin(\omega_d (t-\tau)) \quad (24)$$

It is important to keep in mind that the physical parameters of the IRF matrix (ζ , ω_n , and ω_d) are determined by the baseline model. It should also be pointed out that Equation (24) applies to an underdamped MDOF system ($\zeta < 1$) with no rigid body modes; a reasonable assumption for a cantilevered beam.

D. INTEGRAL EQUATION FORMULATION FOR STRUCTURAL MODIFICATION

When a structural modification has been made, the convolution integral will model the modified response $\{x^*(t)\}$:

$$\{x^*(t)\} = \int_0^t [h(t-\tau)] \{f(\tau)\} d\tau \quad (25)$$

where $\{f(\tau)\}$ is the sum of the externally applied excitations and the new force imposed on the system as a result of the system modification and is as follows:

$$\{f(\tau)\} = \{f^e(\tau)\} - \{f^*(\tau)\} \quad (26)$$

where $\{f^e(t)\}$ is the vector of externally applied excitations, and $\{f^*(t)\}$ is the change in force due to structural modifications and can be written as:

$$\{f^*(\tau)\} = \{[\Delta M]\{\ddot{x}(\tau)\} + [\Delta C]\{\dot{x}(\tau)\} + [\Delta K]\{x(\tau)\}\} \quad (27)$$

Substituting Equation (21) and Equation (22) into Equation (20), and assuming there are no mass or damping modifications, the modified convolution integral becomes:

$$\{x^*(t)\} = \int_0^t [h(t-\tau)] [\{f^e(\tau)\} - [\Delta K]\{x^*(\tau)\}] d\tau \quad (28)$$

$$\{x^*(t)\} = \int_0^t [h(t-\tau)] \{f^e(\tau)\} d\tau - \int_0^t [h(t-\tau)] [\Delta K] \{x^*(\tau)\} d\tau \quad (29)$$

Recognizing that the first term is Equation (9), the baseline response of the unmodified system, Equation (23) can be reduced to:

$$\{x^*(t)\} = \{x(t)\} - \int_0^t [h(t-\tau)][\Delta K]\{x^*(\tau)\} d\tau \quad (30)$$

This equation takes the form of Equation (2), a VIE of the second kind. Recall that the unmodified system parameters are used to construct the IRF matrix and therefore it is clear that the modified system response only requires the baseline system response and its parameters, and the structural modification. Though this paper focuses on strictly stiffness modifications, the analysis can be extended to mass and damping modifications, or any combination of structural modifications. This extends the analysis to solving integral equations with first and second order terms.

E. STABILITY OF THE INTEGRAL EQUATION FORMULATION FOR STRUCTURAL SYNTHESIS

In order to understand the stability of the integral equation formulation for structural synthesis, one only needs to study the coarse solution derivation for the adaptive method when applied to the structural synthesis Equation (25). Since this is a VIE of the second kind, the process outlined by Gordis and Neta [3] can be followed. Since this derivation is not trivial, it has been included as Appendix A.

Starting with Equation (25):

$$\{x^*(t)\} = \{x(t)\} - \int_0^t [h(t-\tau)][\Delta K]\{x^*(\tau)\} d\tau$$

and following the steps as outlined by [3], the coarse solution of Equation (25) is:

$$\left([I] + \frac{1}{2} \delta_j [h]_{jj} [\Delta K] \right) \{x^*\}_j = - \sum_{i=0}^{j-1} \frac{1}{2} (\delta_{i+1} + \delta_i) [h]_{ji} [\Delta K] \{x^*\}_i + \{x\}_j \quad (31)$$

where δ is the time-step for a given interval. Also from Equation (19), it is seen that $[h]_{jj} = 0$ (when $\sin(t-\tau) = 0$). Equation (26) now becomes:

$$\{x^*\}_j = -\sum_{i=0}^{j-1} \frac{1}{2}(\delta_{i+1} + \delta_i)[h]_{ji}[\Delta K]\{x^*\}_i + \{x\}_j \quad (32)$$

Equations (26) and (27) give the basis for the discussion of stability. The structure used in this analysis is an aluminum cantilevered beam. The beam is developed as a finite element model in which a stiffness modification can be made to any of the elements. An excellent discussion concerning the construction of a finite element model for a cantilevered beam is given by Kwon and Bang [10]. The key detail relevant to the discussion of stability is the element stiffness matrix. As shown in [10], the element stiffness matrix is as follows:

$$[K^e] = \frac{EI}{l^3} \begin{bmatrix} 12 & 6l & -12 & 6l \\ 6l & 4l^2 & -6l & 2l^2 \\ -12 & -6l & 12 & -6l \\ 6l & 2l^2 & -6l & 4l^2 \end{bmatrix} \quad (33)$$

The element stiffness modification to the structure can be represented as:

$$[\Delta K] = \eta [K^e] \quad (34)$$

where η is the percent change of the element stiffness. The parameters of the aluminum beam are: Young's Modulus (E) = 69 GPa (10^6 psi), length (l) = 3.05 m (120 in), depth (d) = 0.3 m (12 in), and width (w) = 0.1 m (4 in). This results in a moment-of-inertia (I) of 0.01 m^3 (576 in^3). When substituting the values into Equation (28), the magnitude of the stiffness change is on the order of 1×10^8 .

Understanding that the coarse solution for the integral formulation for structural synthesis is governed by Equation (27), the kernel is then defined as:

$$[k]_{ij} = -\frac{1}{2}(\delta_{i+1} + \delta_i)[h]_{ji}[\Delta K] \quad (35)$$

The stiffness modification is a fixed value and can be of sufficient magnitude to make the VIE conditionally stable for the given time-step. The only parameter that can be adjusted is choice of the time-step; affecting both $(\delta_i + \delta_{i+1})$ and $[h]_{ji}$

terms sufficiently to return the VIE to stability. This was observed running the code that modelled the cantilevered beam. If the time-step was not sufficiently small for a given stiffness modification, then the solution was unstable. A sufficiently small time-step would give a stable and accurate solution but will increase the computation time.

Trying to determine a sufficiently small time-step for stability is not effective, especially when considering an adaptive time-step method. Returning to Equation (26), it is important to realize that the left hand IRF function matrix will always be zero. A non-zero left hand IRF function would effectively normalize Equation (26) and maintain unconditional stability of the equation. In order to obtain a non-zero term on the left-hand side, the logical step is to take the derivative of the synthesis equation.

F. DERIVATIVE OF THE SYNTHESIS EQUATION

The goal of taking the derivative of the synthesis equation is to ensure that the kernel does not equal zero when $(\tau = t)$. The full derivation of the derivative of $\{\dot{x}^*\}$ is shown in Appendix B, but the resulting equation is:

$$\{\dot{x}^*(t)\} = \{\dot{x}(t)\} - \int_0^t [\dot{h}(t-\tau)] [\Delta K] \{x^*(\tau)\} d\tau \quad (36)$$

a VIDE. Now the entries of the derivative of the IRF matrix take the form:

$$\dot{h}_{ij}(t-\tau) = \sum_{p=1}^n (-\phi_i^p \phi_j^p \frac{\omega_n \zeta}{\omega_d} e^{-\omega_n \zeta (t-\tau)} \sin(\omega_d (t-\tau)) + \phi_i^p \phi_j^p e^{-\omega_n \zeta (t-\tau)} \cos(\omega_d (t-\tau))) \quad (37)$$

When $\tau = t$, Equation (32) reduces to:

$$\dot{h}_{ij}(0) = \sum_{p=1}^n \phi_i^p \phi_j^p \quad (38)$$

Or in terms of the system:

$$[\dot{h}(0)] = [\Phi][\Phi]^T \quad (39)$$

Applying the process outlined in [3], the coarse solution of Equation (31) is:

$$\begin{aligned}
& (I + \frac{1}{4}\delta_j^2 [\dot{h}]_{jj} [\Delta K]) \{\dot{x}^*\}_{jj} = \\
& -\sum_{i=0}^{j-1} \frac{1}{2} (\delta_{i+1} + \delta_i) [\dot{h}]_{ji} [\Delta K] \{x^*\}_i - \frac{1}{2} \delta_j [\dot{h}]_{jj} [\Delta K] (\sum_{i=0}^{j-2} \frac{1}{2} \delta_{i+1} (\{\dot{x}^*\}_i + \{\dot{x}^*\}_{i+1})) \quad (40) \\
& -\frac{1}{4} \delta_j^2 [\dot{h}]_{jj} [\Delta K] \{\dot{x}^*\}_{j-1} - \frac{1}{2} \delta_j [\dot{h}]_{jj} [\Delta K] \{x^*\}_0 + \{\dot{x}\}_j
\end{aligned}$$

The full derivation of Equation (35) can be found in Appendix C. There are a few points to notice:

- 1) Equation (36) is substantially more complicated than Equation (26).
- 2) Equation (36) contains both $\{x^*\}$ and $\{\dot{x}^*\}$ terms.
- 3) The left-hand side of Equation (36) contains a non-zero derivative of the IRF matrix. Substituting Equation (34) into the left-hand side of Equation (35), the left-hand side of Equation (35) can be re-written as:

$$(I + \frac{1}{4}\delta_j^2 [\dot{h}]_{jj} [\Delta K]) \{\dot{x}^*\}_{jj} = \{\dot{x}^*\}_{jj} (I + \frac{1}{4}\delta_j^2 [\Phi][\Phi]^T [\Delta K]) \quad (41)$$

The first challenge is to determine if Equation (35) can be solved in its current form with both $\{x^*\}$ and $\{\dot{x}^*\}$ terms present. Using the trapezoidal rule, $\{\dot{x}^*\}_j$ can be written as follows:

$$\{\dot{x}^*\}_j = \frac{\delta_1}{2} (\{\dot{x}^*\}_0 + \{\dot{x}^*\}_1) + \frac{\delta_2}{2} (\{\dot{x}^*\}_1 + \{\dot{x}^*\}_2) + \dots + \frac{\delta_j}{2} (\{\dot{x}^*\}_{j-1} + \{\dot{x}^*\}_j) + \{x^*\}_0 \quad (42)$$

Due to the iterative nature of the algorithm, the current modified displacement, $\{x^*\}_j$, can be solved as soon as the velocity, $\{\dot{x}^*\}_j$, for the given time step is solved. Furthermore, the previously calculated modified displacement, $\{x^*\}_{j-1}$, and previously calculated modified velocity, $\{\dot{x}^*\}_{j-1}$, can be used to calculate the current modified displacement. This becomes clear after a few iterations:

$$\begin{aligned}
\{x^*\}_1 &= \frac{\delta_1}{2} (\{\dot{x}^*\}_0 + \{\dot{x}^*\}_1) + \{x^*\}_0 \\
\{x^*\}_2 &= \frac{\delta_1}{2} (\{\dot{x}^*\}_0 + \{\dot{x}^*\}_1) + \frac{\delta_2}{2} (\{\dot{x}^*\}_1 + \{\dot{x}^*\}_2) + \{x^*\}_0 \\
\{x^*\}_2 &= \{x^*\}_1 + \frac{\delta_2}{2} (\{\dot{x}^*\}_1 + \{\dot{x}^*\}_2) \\
\{x^*\}_3 &= \frac{\delta_1}{2} (\{\dot{x}^*\}_0 + \{\dot{x}^*\}_1) + \frac{\delta_2}{2} (\{\dot{x}^*\}_1 + \{\dot{x}^*\}_2) + \frac{\delta_3}{2} (\{\dot{x}^*\}_2 + \{\dot{x}^*\}_3) + \{x^*\}_0 \\
\{x^*\}_3 &= \{x^*\}_2 + \frac{\delta_3}{2} (\{\dot{x}^*\}_2 + \{\dot{x}^*\}_3) \\
&\vdots \\
\{x^*\}_j &= \{x^*\}_{j-1} + \frac{\delta_j}{2} (\{\dot{x}^*\}_{j-1} + \{\dot{x}^*\}_j)
\end{aligned}$$

This highlights one of the benefits of evaluating the derivative of the synthesis equation. Even though the resulting algorithm is far more complicated than the original synthesis equation, not only is the velocity obtained, the modified displacement is also obtained.

The next question to be answered is to whether the non-zero derivative of the IRF matrix on the left-hand side returns the VIDE to unconditional stability.

The term $\left([I] + \frac{1}{4} \delta_j^2 [\dot{h}]_{jj} [\Delta K] \right)$ is analogous to a scalar in a SDOF problem.

Performing the appropriate operations to solve for $\{\dot{x}^*\}_j$ is also analogous to dividing the equation by the scalar in a SDOF problem. This normalizes the equation at each time-step, essentially reducing the effect that the stiffness modification has on the system and maintaining unconditional stability.

Determining the stability of integral equations is not a trivial task and is usually addressed on a case-by-case basis. In order to determine whether the derivative of the synthesis equation was unconditionally stable, a range of time-steps was tested. If the system was still conditionally stable, at a large time-step the system would produce an unstable result. Running the program with an initial time-step of $1.0 \times 10^{-3} \text{ sec}$ produced a stable, but inaccurate solution that closely

modelled the unmodified system response. The step-size was reduced by a magnitude of 10 until the time-step of $1 \times 10^{-5} \text{ sec}$ was reached. At this time-step, the solution closely followed the exact solution to the modified system response. Not only is the system unconditionally stable, but it produces accurate results for the modified system with the appropriate time-step.

III. MODELS AND RESULTS

A. GENERAL PROCESS

The following models have been solved using the adaptive solution procedure:

- 1) A SDOF mass-spring system with an externally applied step excitation, subjected to a stiffness modification.

$$\text{mass } (m) = 1kg \text{ (2.2 lb)}, \text{ spring constant } (k) = 100N / m \left(6.9 \frac{lb}{ft} \right)$$

- 2) A SDOF mass-spring system with an externally applied periodic excitation, subjected to a stiffness modification.

$$\text{mass } (m) = 1kg \text{ (2.2 lb)}, \text{ spring constant } (k) = 100N / m \left(6.9 \frac{lb}{ft} \right)$$

- 3) A two DOF mass spring system with an externally applied step excitation, subjected to a stiffness modification to the second spring.

$$\text{mass } (m) = 1kg \text{ (2.2 lb)}, \text{ spring constant } (k) = 100N / m \left(6.9 \frac{lb}{ft} \right)$$

- 4) A generalized MDOF cantilevered aluminum beam with an externally applied step excitation, subjected to a stiffness modification to an arbitrary beam element. Recall this is the model in which conditional stability has been observed.
- 5) The same model as 4, but the derivative of the synthesis equation has been analyzed in order to determine if unconditional stability is observed.

The program to the adaptive method has three distinct phases:

- 1) Programming algorithms for the coarse, half, and fine approximations as derived from Gordis and Neta [3].
- 2) Determining the error between the coarse and fine approximation.
- 3) Halving or doubling the time-step as determined by the error.

Since the algorithms of [3] were derived for the generalized VIE of the second kind, the algorithms had to be altered for the specific model. In order to ensure that the algorithms were correct, a non-adaptive approach was implemented. Since the exact solution for all models is known, the results of the course, half, and fine approximations were calculated and individually compared against the exact solution. Once these algorithms were properly programmed, then the error assessment and adaptive time-step adjustment phases were implemented. These algorithms did not significantly deviate from model-to-model.

As previously mentioned, the adaptive solution for the cantilever beam exhibited conditional stability while evaluating the coarse approximation. At this point in the analysis, the coarse approximation was not yet adaptive and is simply a standard trapezoid rule. Once conditional stability had been exhibited in the coarse approximation, the half and fine approximations were not developed and the derivative of the synthesis equation was pursued. In developing the coarse approximations to the derivative, it became clear that though this had produced an unconditionally stable and accurate solution, the algorithm is limited by computational efficiency. As a result, the half and fine approximations were not developed for this model as well.

B. SDOF MASS-SPRING SYSTEM WITH EXTERNALLY APPLIED STEP FUNCTION SUBJECTED TO A STIFFNESS MODIFICATION

The physical representation of this model is given in Figure 1. The stiffness

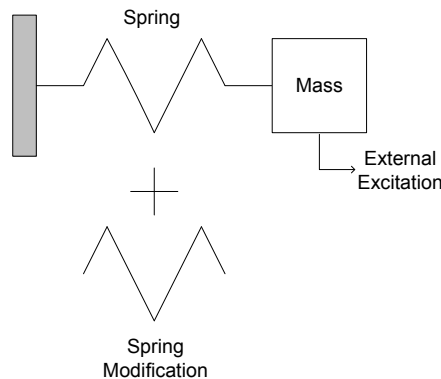


Figure 1. SDOF mass-spring with stiffness modification

modification is varied in the following intervals: 0%, 10%, 25%, 50%, 75%, and 100% at an initial time-step of 0.01 second, and a maximum error between coarse and fine approximations of 1×10^{-4} . Table 1 gives the accuracy, total number of calculations, and time to complete for each stiffness modification. The plots of modified displacement and step size for 10%, 50% and 100% have been included to show the variance in system response as the stiffness modification is increased.

Stiffness Modification (%)	Greatest Error (%)	Least Error (%)	Total number of calculations	Time to complete(sec)
0	0.9	0.0	7	0.16
10	1.3	0.0	902	3.7
25	2.6	0.0	1184	5.0
50	4.5	0.0	1657	7.1
75	6.2	0.0	1803	7.9
100	7.6	0.0	1913	8.5

Table 1. Response to stiffness modifications for SDOF mass-spring system subjected to externally applied step excitation.

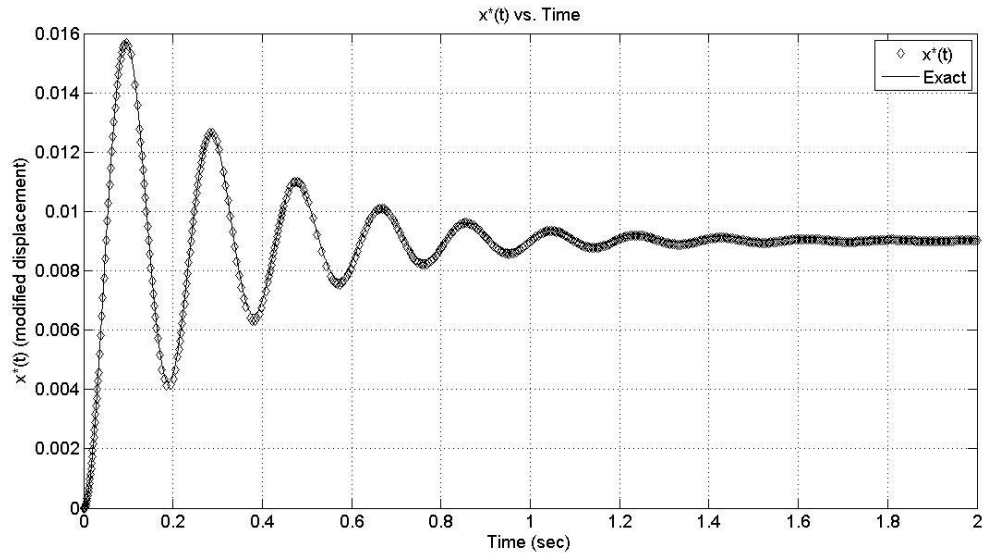


Figure 2. Modified response for a SDOF mass-spring with externally applied step excitation subjected to 10% stiffness modification.

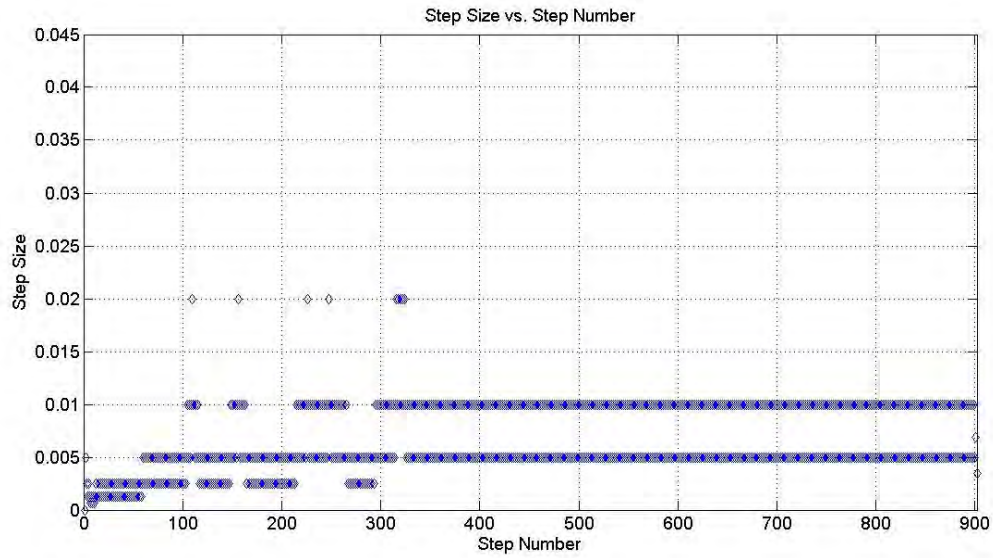


Figure 3. Step size for a SDOF mass-spring with externally applied step excitation subjected to 10% stiffness modification.

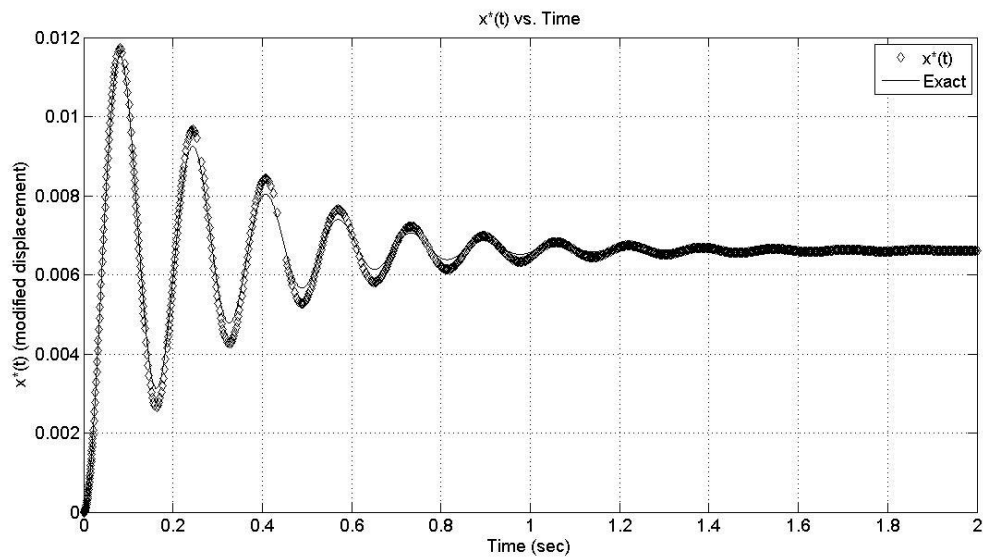


Figure 4. Modified response for a SDOF mass-spring with externally applied step excitation subjected to 50% stiffness modification.

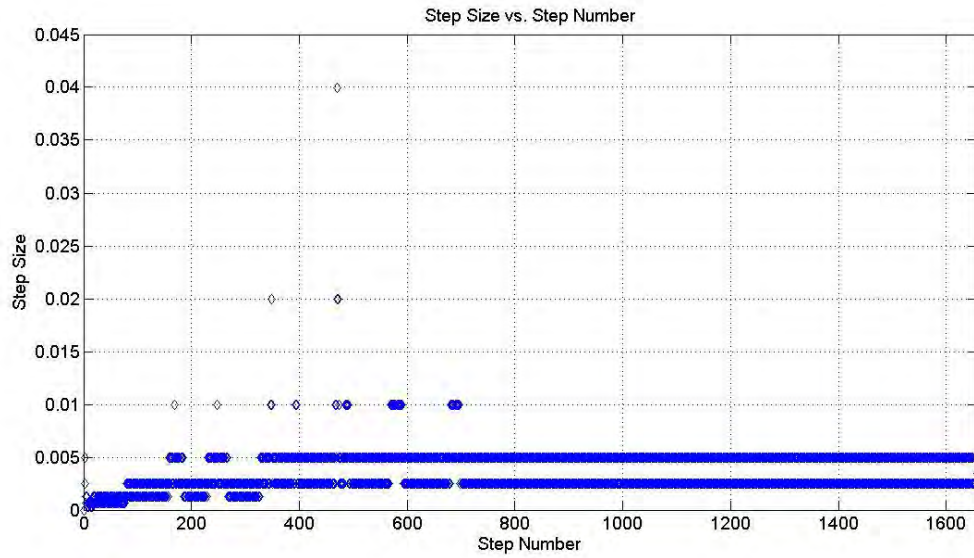


Figure 5. Step size for a SDOF mass-spring with externally applied sinusoidal excitation subjected to 50% stiffness modification.

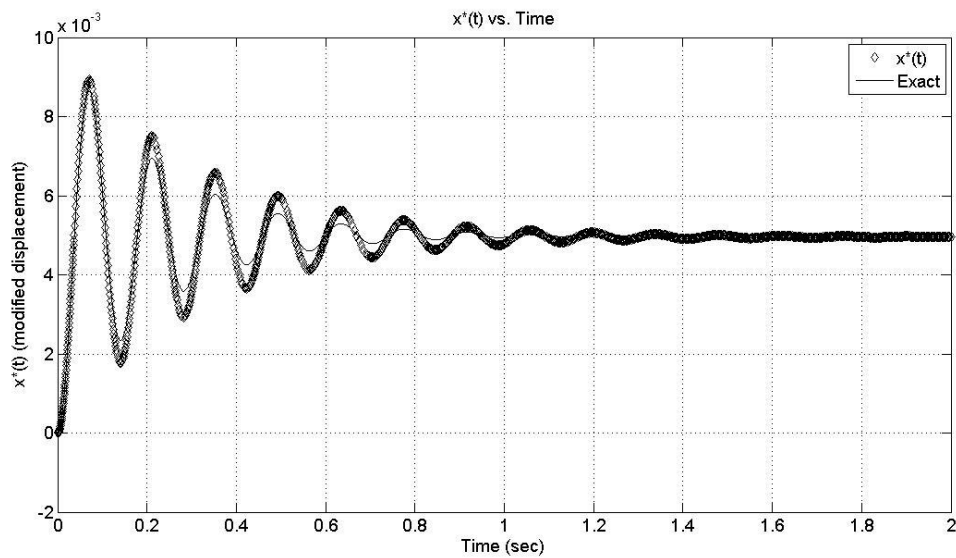


Figure 6. Modified response for a SDOF mass-spring with externally applied step excitation subjected to 100% stiffness modification.

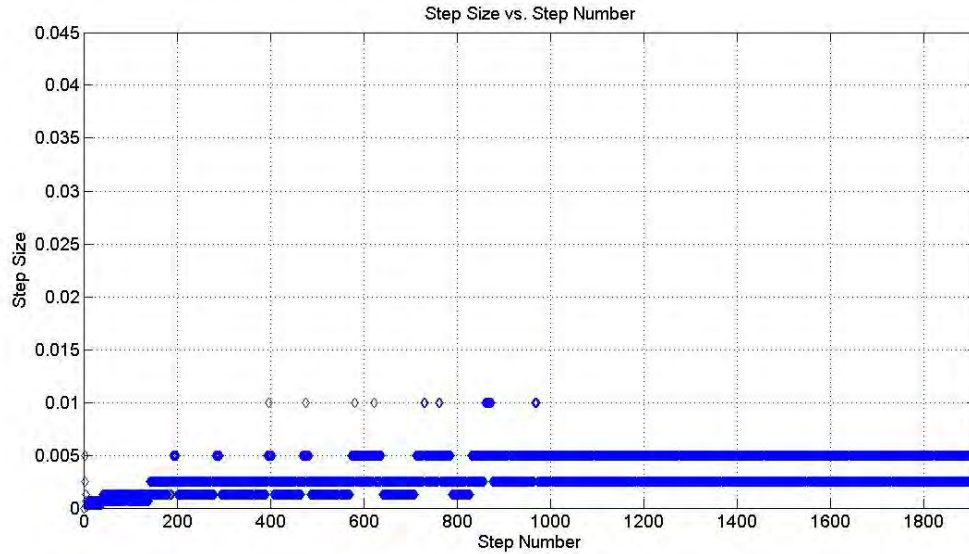


Figure 7. Step size for a SDOF mass-spring with externally applied sinusoidal excitation subjected to 50% stiffness modification.

Comparing Figures 2, 4, and 6 show that the lower the stiffness modification, the more accurate the approximation to the exact solution. This is also evident in the maximum error given by Table 1. Also Figures 3, 5, and 7 shows that the lower the stiffness modification, the difference between the time-steps are greater. Consequently, the number of iterations required to approximate the response is lower than that for the higher stiffness modifications. It is important to keep in mind that accuracy is not just determined by the stiffness modification, but is determined by the time-step as well. A higher stiffness modification requires a smaller time-step for accuracy is dependent on the ratio of the stiffness modification to the time-step.

It should be noted from Table 1 with a 0% stiffness modification the error is very low, and the time-step was doubled every iteration. Though the system was approximated in seven time-steps, the result is not a good approximation of the system response. Though Gordis and Neta [3] place a lower limit on the time-step as a termination requirement in case a function cannot be approximated, for systems with low modifications an upper limit is needed in order to accurately represent the system response.

Finally, the measure of efficiency needs to be addressed. The ideal measure of efficiency is the combination of speed and accuracy. Clearly, speed and the number of time-steps are directly proportional as evidenced by Table 1. A non-adaptive trapezoidal method from zero to two seconds at a time-step of 0.01 seconds would require 200 iterations to approximate the system response. However, accuracy may be lost when using a non-adaptive trapezoidal method. Table 1 shows that for a stiffness modification of 10% or higher, the number of iterations is greater than that required from the non-adaptive approach. Efficiency is clearly lost in terms of speed with an adaptive approach, but a higher degree of accuracy may be preserved.

There are many variables that have to be taken into account when discussing efficiency. Table 1 gives the results for the most obvious variable that can affect efficiency, stiffness modification. Other variables to consider may be, but not limited to, maximum error between the coarse and fine approximations, and the acceptable error between the approximation of the modified displacement and known or measured data (for this discussion the exact solution). For this particular model it is shown that the adaptive approach can give reasonably accurate results over a range of stiffness modifications. It is up to the analyst to consider the usefulness of this method for their particular model.

C. SDOF MASS-SPRING SYSTEM WITH EXTERNALLY APPLIED PERIODIC FUNCTION SUBJECTED TO A STIFFNESS MODIFICATION

The physical representation of this model is given by Figure 1 where the external excitation is now a sinusoidal forcing function. The stiffness modification for this system is varied in the following intervals: 0%, 10%, 25%, 50%, 75%, and 100% at an initial time-step of 0.01 second, and a maximum error between coarse and fine approximations of 1×10^{-4} . The results of these modifications are presented in Table 2. The plots of modified displacement and step size for 10%, 50% and 100% have been included to show the variance in system response as the stiffness modification is raised. Many of the same observations for this

system follow the observations made in the previous section with the system subjected to an external step excitation.

Stiffness Modification(%)	Greatest Error (%)	Least Error (%)	Total number of calculations	Time to complete(sec)
0	0.0	0.0	7	0.1
10	0.8	0.0	638	1.9
25	1.6	0.0	812	2.8
50	3.1	0.0	1119	3.9
75	3.8	0.0	1277	4.5
100	4.6	0.0	1339	4.7

Table 2. Response to stiffness modifications for SDOF mass-spring system subjected to externally applied sinusoidal excitation.

An interesting point to note is that the error obtained from this system is smaller than that for the system subjected to external step excitation as the stiffness modification is increased. This result would make sense if the corresponding number of calculations was higher for the periodic function. Since it is not, the reason for the lower error is not understood and needs to be investigated further.

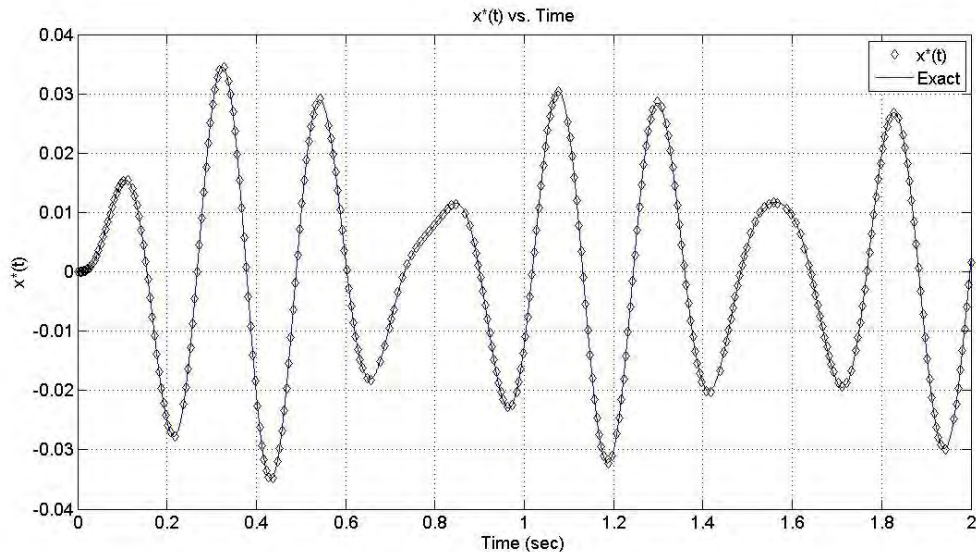


Figure 8. Modified response for a SDOF mass-spring with externally applied sinusoidal excitation subjected to 10% stiffness modification.

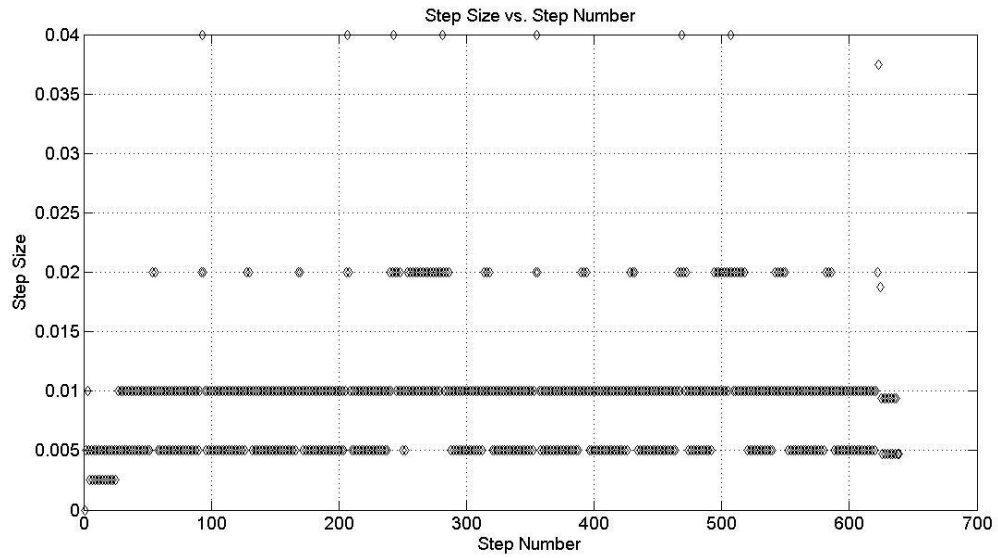


Figure 9. Step size for a SDOF mass-spring with externally applied step excitation subjected to 10% stiffness modification.

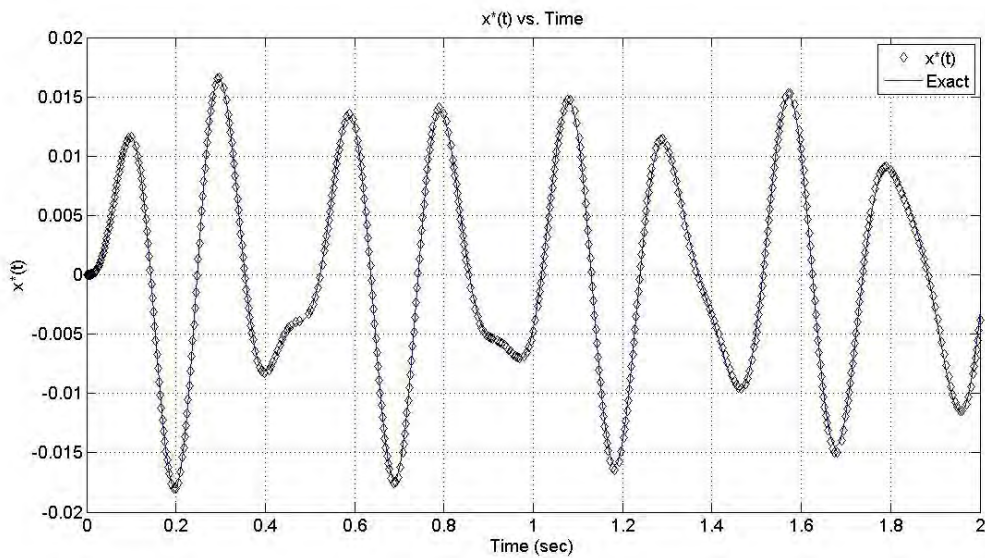


Figure 10. Modified response for a SDOF mass-spring with externally applied sinusoidal excitation subjected to 50% stiffness modification.

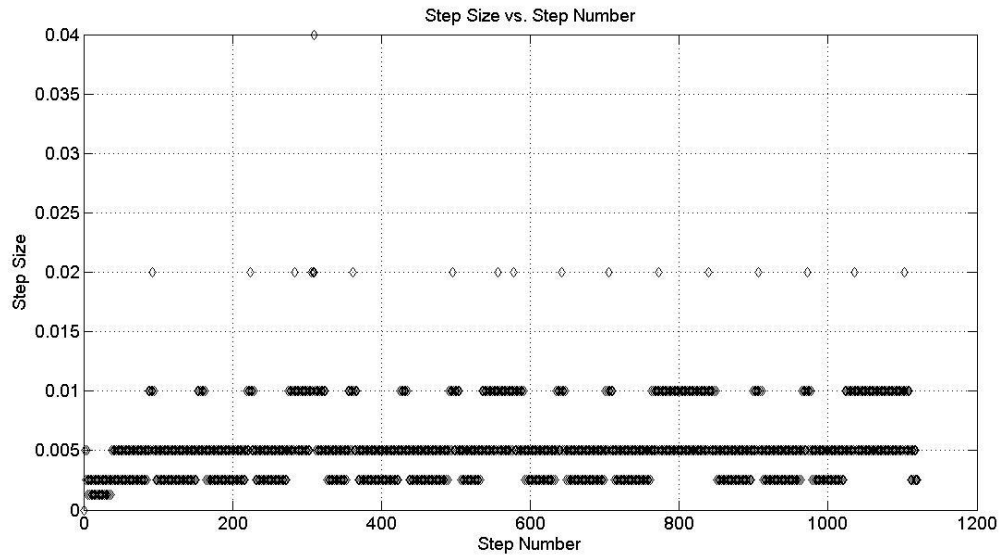


Figure 11. Step size for a SDOF mass-spring with externally applied step excitation subjected to 50% stiffness modification.

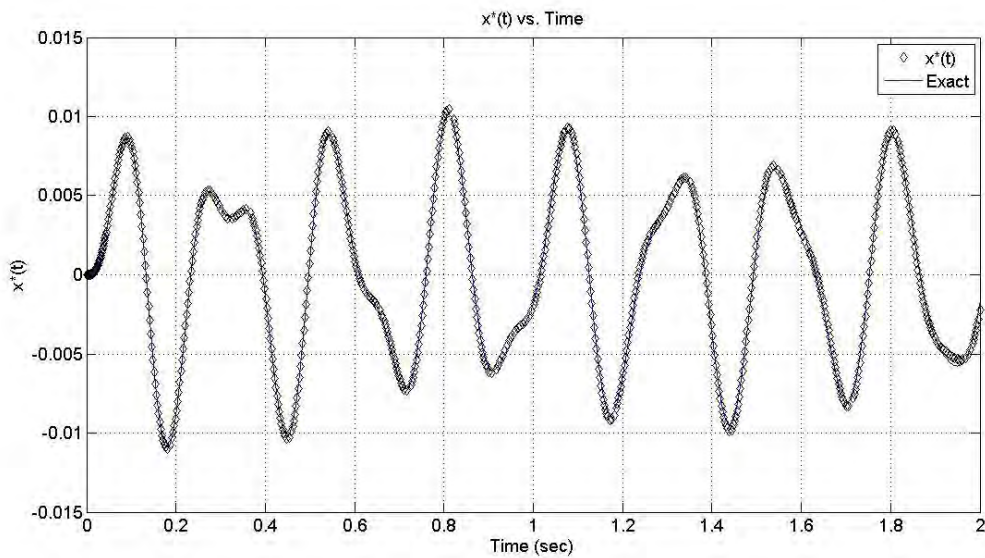


Figure 12. Modified response for a SDOF mass-spring with externally applied sinusoidal excitation subjected to 100% stiffness modification.

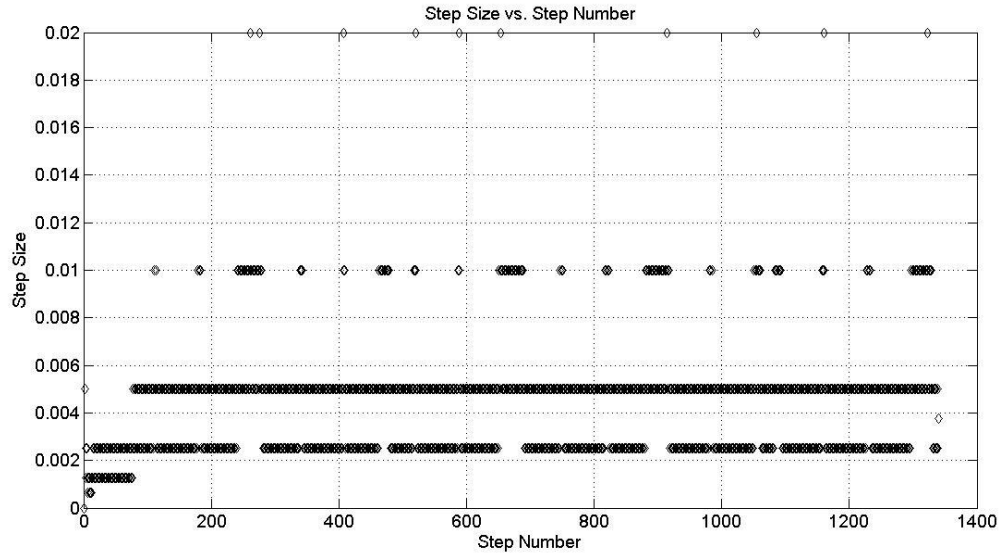


Figure 13. Step size for a SDOF mass-spring with externally applied step excitation subjected to 100% stiffness modification.

D. MDOF MASS-SPRING SYSTEM WITH EXTERNALLY APPLIED STEP FUNCTION SUBJECT TO A STIFFNESS MODIFICATION

The physical representation of this model is given by Figure 14.

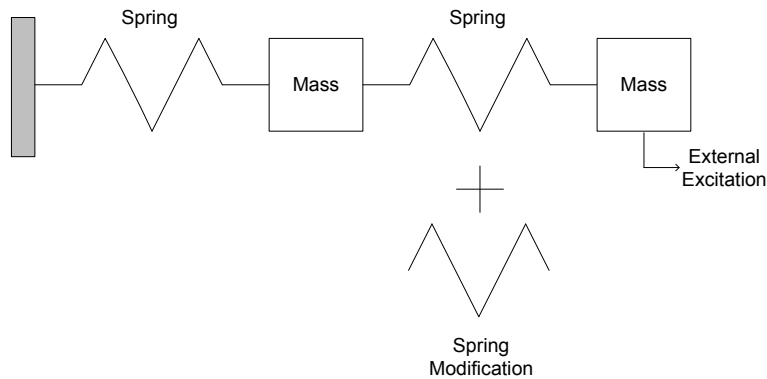


Figure 14. Two DOF mass-spring system subjected to stiffness modification.

The stiffness modification is applied to the second spring, and the external excitation is a step function applied to the second mass of the system. Stiffness modifications of 0%, 25%, 50%, 75%, and 100% are made at an initial time-step

of 0.001 second, and a maximum error of 1×10^{-4} between the coarse and fine approximations. The results can be found in Table 3.

Stiffness Modification (%)	Greatest Error (%) (Mode 1)	Greatest Error (%) (Mode 2)	Total number of calculations	Time to complete(sec)
0	0.5	0.5	11	0.08
10	1.7	1.2	3899	91
25	2.1	1.2	5590	181
50	1.7	1.0	7765	343
75	2.0	1.1	8937	450
100	1.8	1.2	9507	505

Table 3. Response to stiffness modifications for a two DOF mass-spring system subjected to externally applied step excitation.

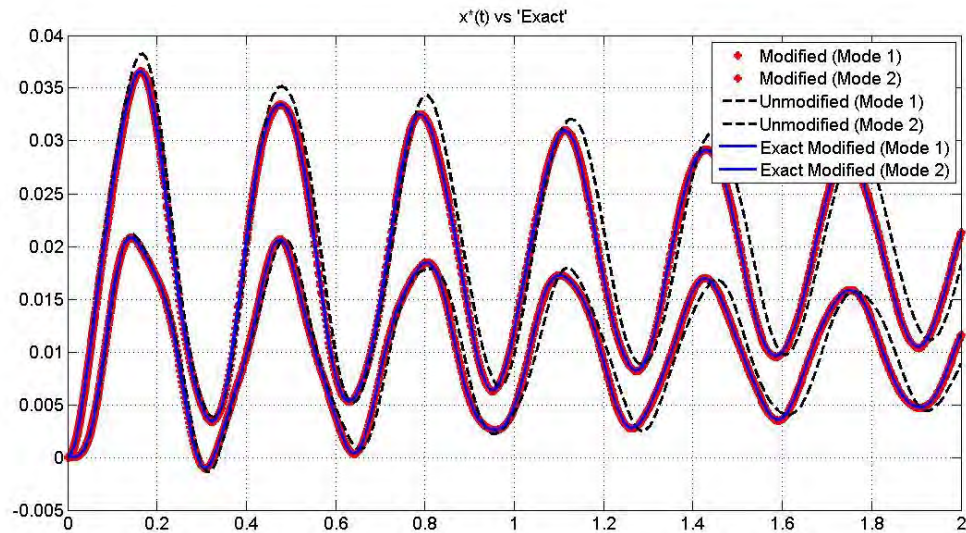


Figure 15. Modified response for a two DOF mass-spring with externally applied step excitation subjected to 10% stiffness modification.

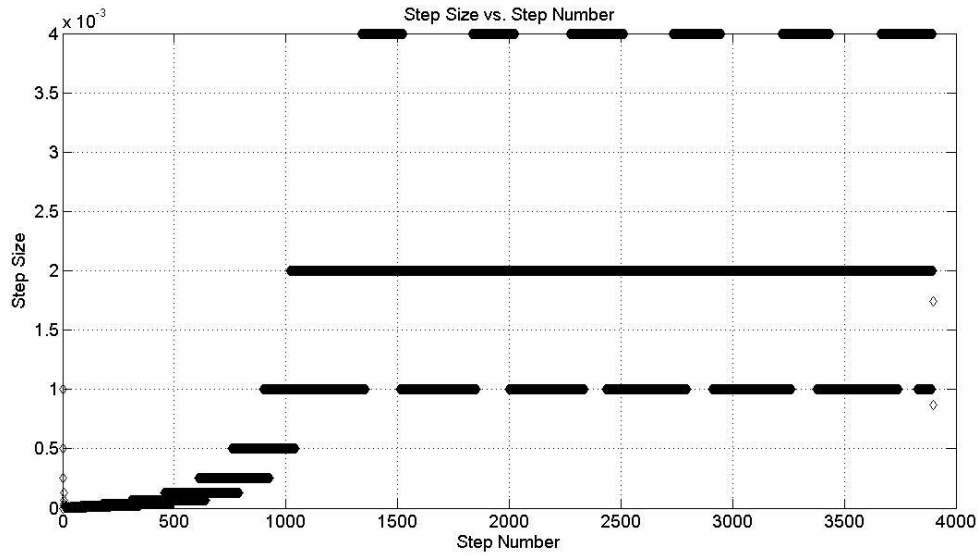


Figure 16. Step size for a two DOF mass-spring with externally applied step excitation subjected to 10% stiffness modification

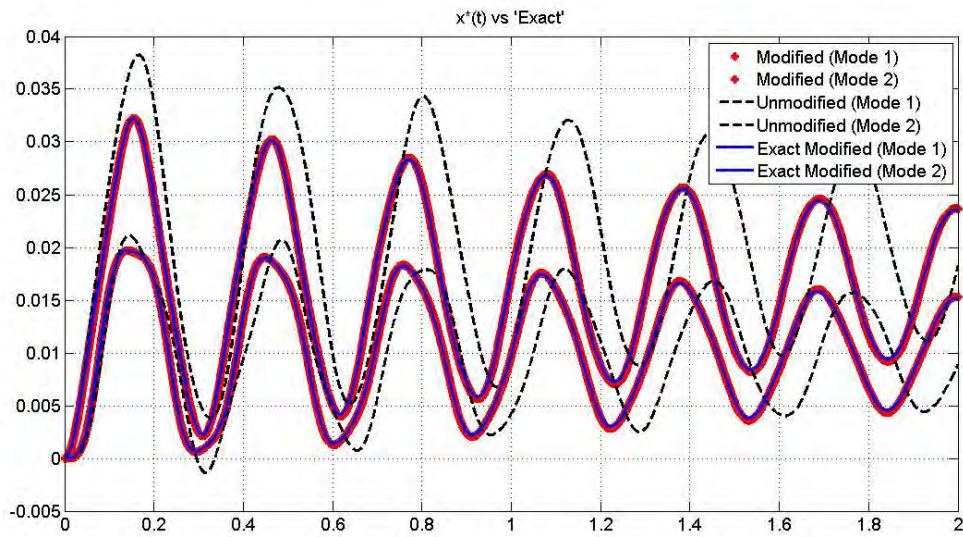


Figure 17. Modified response for a two DOF mass-spring with externally applied step excitation subjected to 10% stiffness modification.

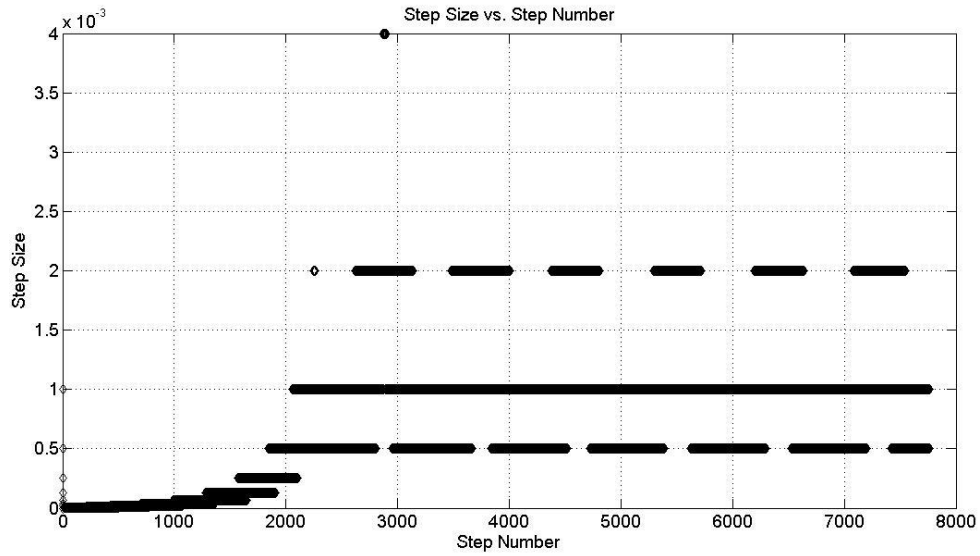


Figure 18. Step size for a two DOF mass-spring with externally applied step excitation subjected to 50% stiffness modification.

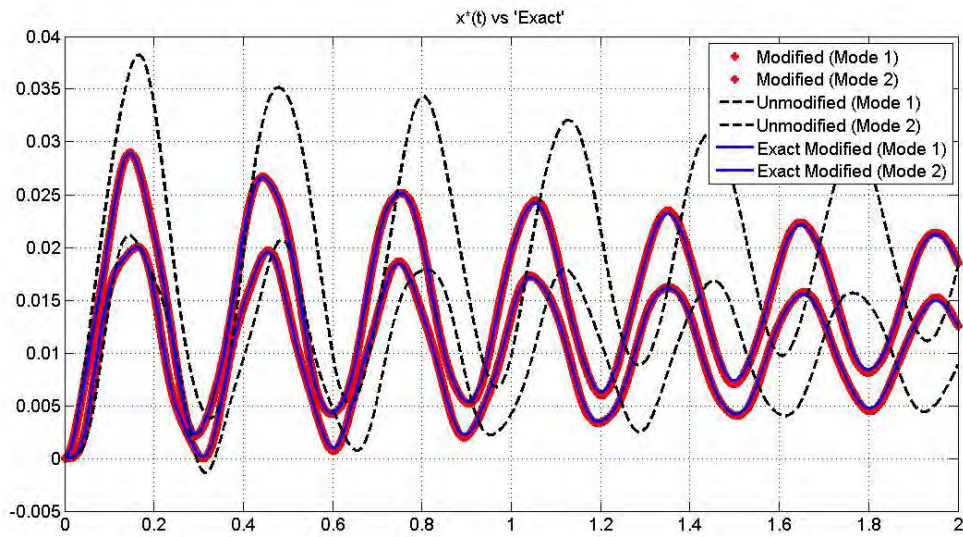


Figure 19. Modified response for a two DOF mass-spring with externally applied step excitation subjected to 100% stiffness modification.

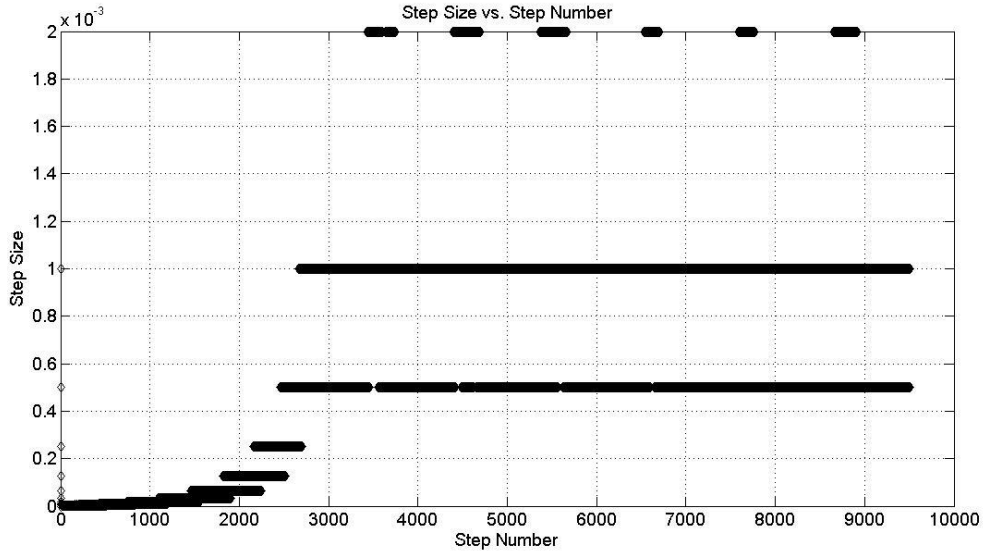


Figure 20. Step size for a two DOF mass-spring with externally applied step excitation subjected to 50% stiffness modification.

From Figures 15, 17, 19, and Table 3, it is immediately apparent that this approximation has a very low error and remains relatively constant as the stiffness modification is increased. However, the number of calculations per stiffness modification goes up as well. This implies a smaller time step, and therefore accuracy is preserved through time-step reduction. This can be seen in Figures 16, 18, and 20 where the time-steps of Figures 18 and 20 are essentially half the value of the time-steps of Figure 16.

E. MDOF CANTILEVERED ALUMINUM BEAM WITH AN EXTERNALLY APPLIED STEP EXCITATION, SUBJECTED TO A STIFFNESS MODIFICATION TO AN ARBITRARY BEAM ELEMENT

This model (Figure 21) is the most important model of this work because it is not only the first model to attempt to apply the adaptive method to a physical structure, but it also the first model to exhibit conditional stability. As mentioned in the Introduction, the magnitude of the stiffness modification drives the integral equation to instability. It should be noted that stability was preserved in the previous models because the parameters of the system were sufficiently small. Once conditional stability was observed in this model, and the reason

determined, the parameters of the SDOF models were raised to the magnitude of the cantilevered beam. The programs were not able to approximate a solution and terminated due to meeting the minimum time-step termination requirement.

Initially the goal of this paper was to apply the adaptive method to the transient analysis of a structure. Once conditional stability was exhibited using the non-adaptive coarse approximation, the goal then became to find a means to restore stability to the approximation. As a result, the half solution and fine approximation algorithms were not developed.

Figures that show the instability for stiffness modifications of 14% and 50% are provided along with figures in which appropriate time-steps have been chosen such that accurate approximations have been obtained. The 14% stiffness modification was chosen because below this value the instability is not obvious.

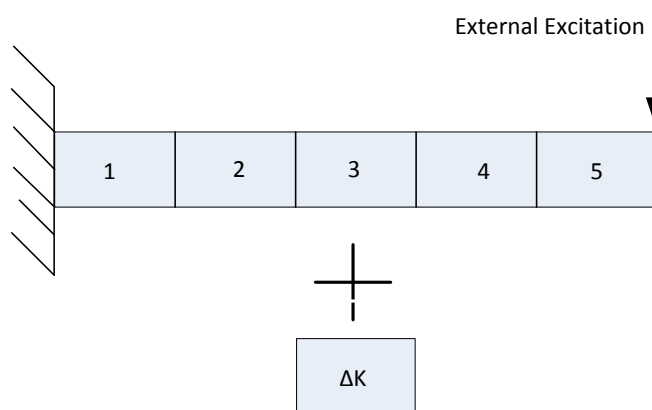


Figure 21. Aluminum cantilevered beam of five elements with an elemental stiffness change to the third element subjected to a step external excitation.

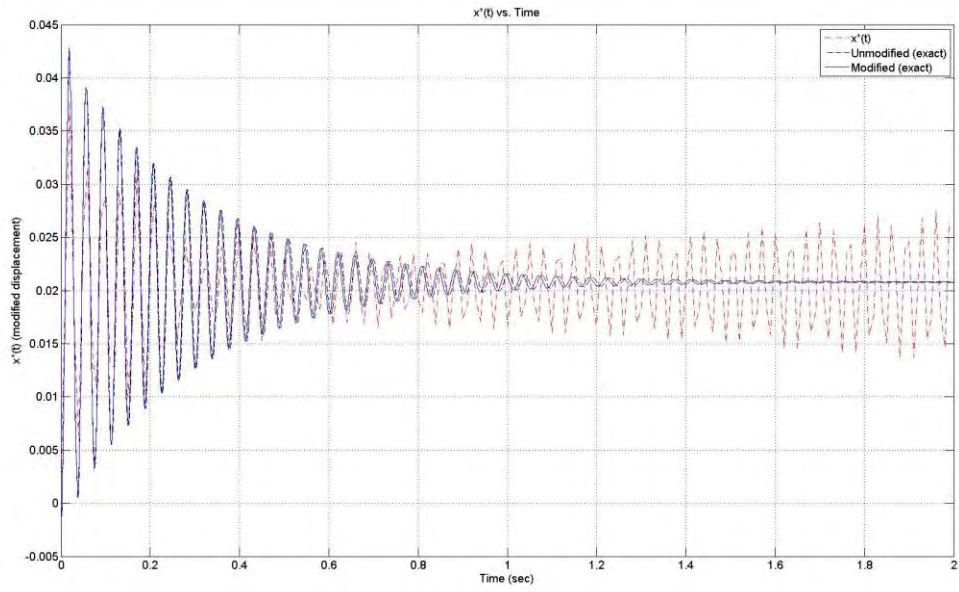


Figure 22. Instability of the aluminum cantilever beam with a stiffness modification of 14% and a time-step of 0.01 second.

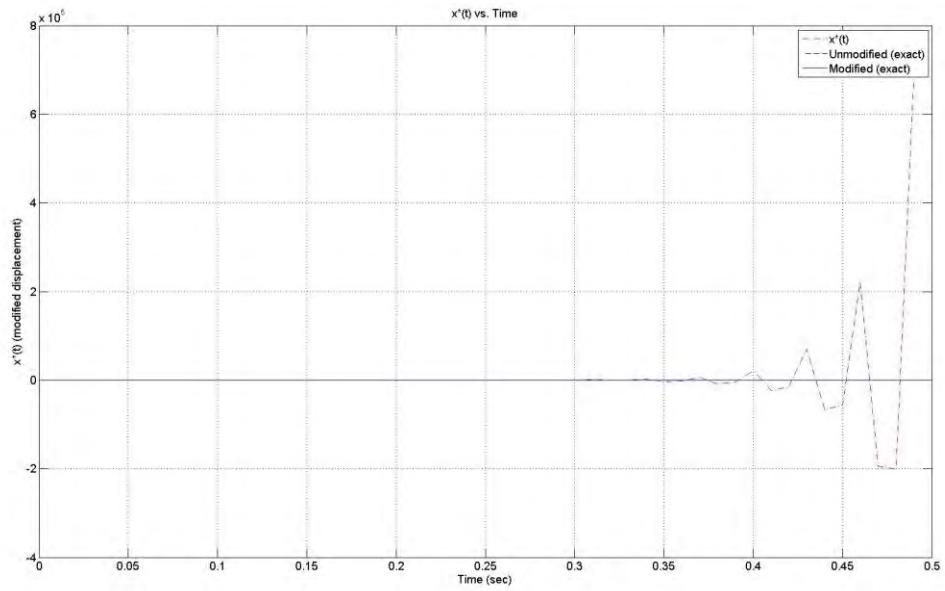


Figure 23. Instability of the aluminum cantilever beam with a stiffness modification of 50% and a time-step of 0.01 second.

Figure 21 and Figure 22 show that the magnitude of the instability is directly related to the magnitude of the stiffness modification. From Equation 35 it is obvious that as the stiffness modification is increased, the magnitude of the kernel increases, and therefore the approximation becomes more unstable. Figure 23 and Figure 24 show that a step-size of sufficient magnitude can restore stability to the approximation. The key observation is that the higher the magnitude of the stiffness modification, a smaller time-step is required to restore stability.

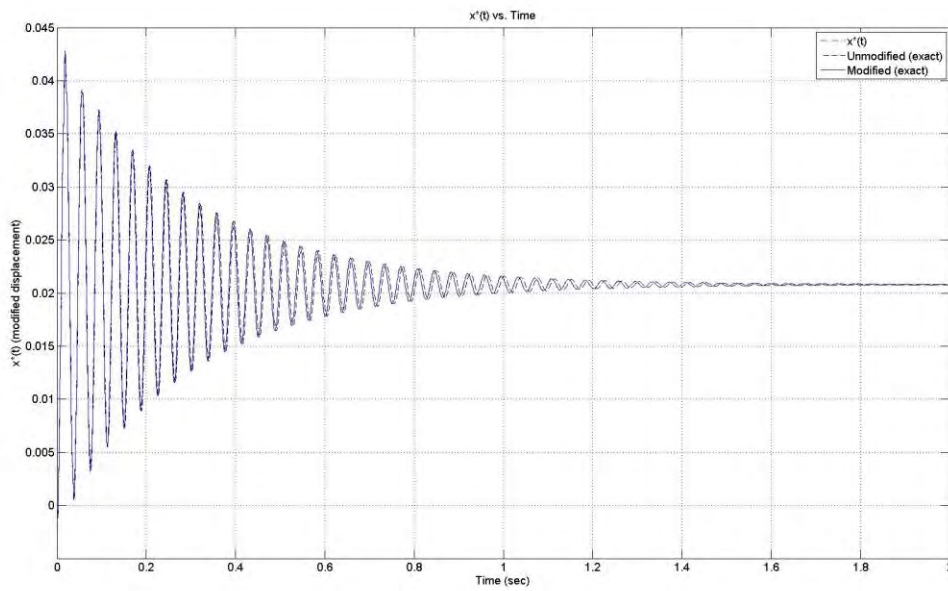


Figure 24. Stability of the aluminum cantilever beam with a stiffness modification of 14% and a time-step of 0.0001 second.

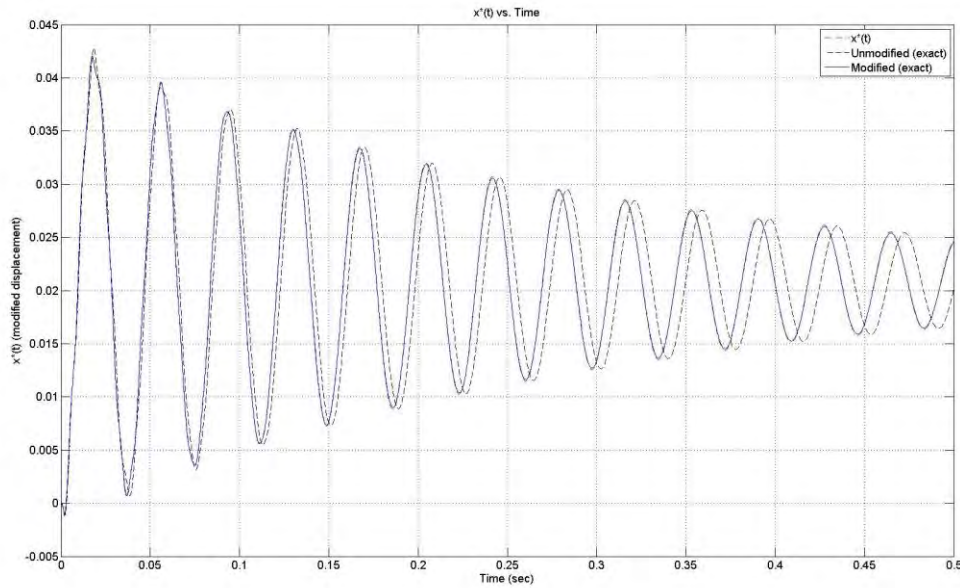


Figure 25. Stability of the aluminum cantilever beam with a stiffness modification of 50% and a time-step of 0.00001 second.

F. THE DERIVATIVE OF THE TRANSIENT STRUCTURAL SYNTHESIS EQUATION FOR THE MDOF CANTILEVERED ALUMINUM BEAM WITH AN EXTERNALLY APPLIED STEP EXCITATION, SUBJECTED TO A STIFFNESS MODIFICATION TO AN ARBITRARY BEAM ELEMENT

The motivation for analyzing the derivative of the synthesis equation is to determine if the derivative is unconditionally stable. Though the derivative results in the velocity of the system response, if unconditionally stable, a numerical integration method can be utilized to determine the position. As previously mentioned, stability of integral equations is a complex study in itself. In order to test stability of the derivative of the synthesis equation, the same time-step that resulted in instability were used. A 100% stiffness modification was implemented to ensure the kernel was of sufficient size to induce instability in the original approximation algorithm, and the time interval was over the time that the system would achieve equilibrium. The system was also run with the same parameters as the non-derivative approximation for comparison purposes.

From Figure 26, it is seen that at 100% stiffness modification, 0.01 second time-step, the system exhibited stability. It should be noted, that the approximation began to trail off over time. Recall in the Theory section that explained in detail the derivative approximation, the modified position was derived using the trapezoidal rule applied to velocities that were derived using the trapezoidal rule. At this time-step, the error is large and can accumulate quickly. Any accuracy the solution may have had is lost quickly.

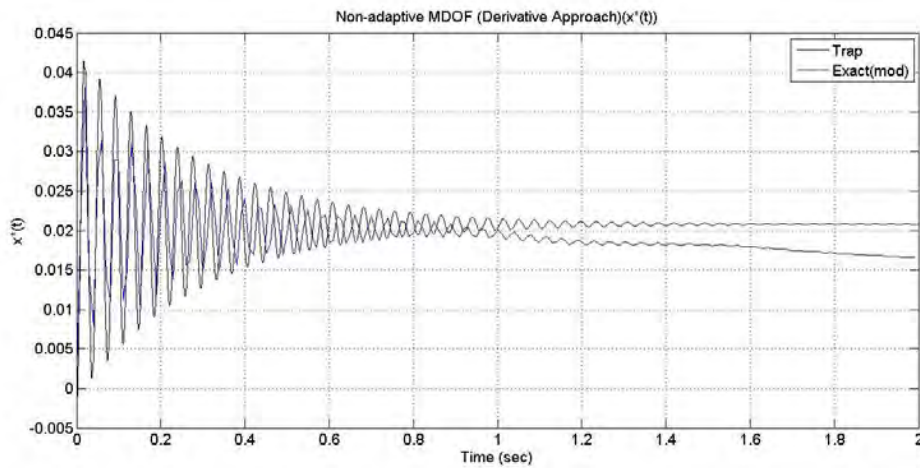


Figure 26. MDOF derivative approximation with a 100% stiffness modification and a 0.01 second time-step exhibiting stability.

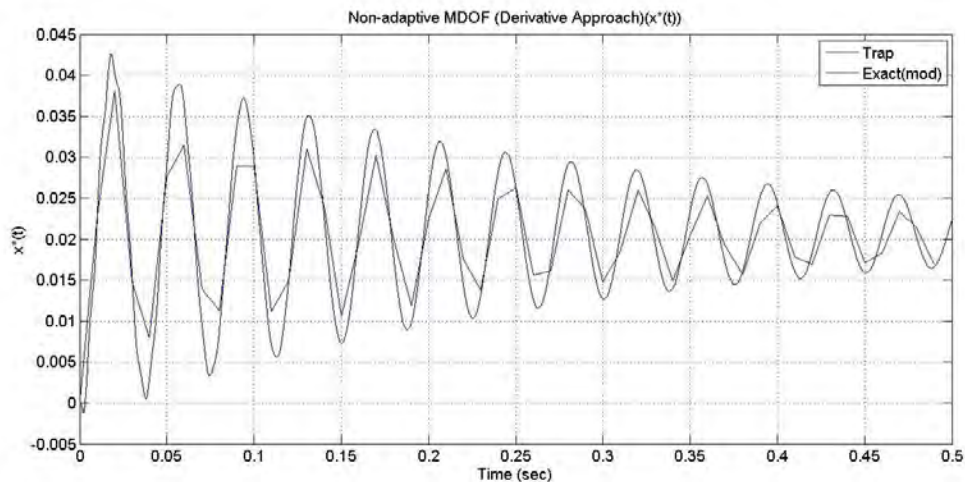


Figure 27. MDOF derivative approximation 14% elemental stiffness modification and a time-step of 0.01 second.

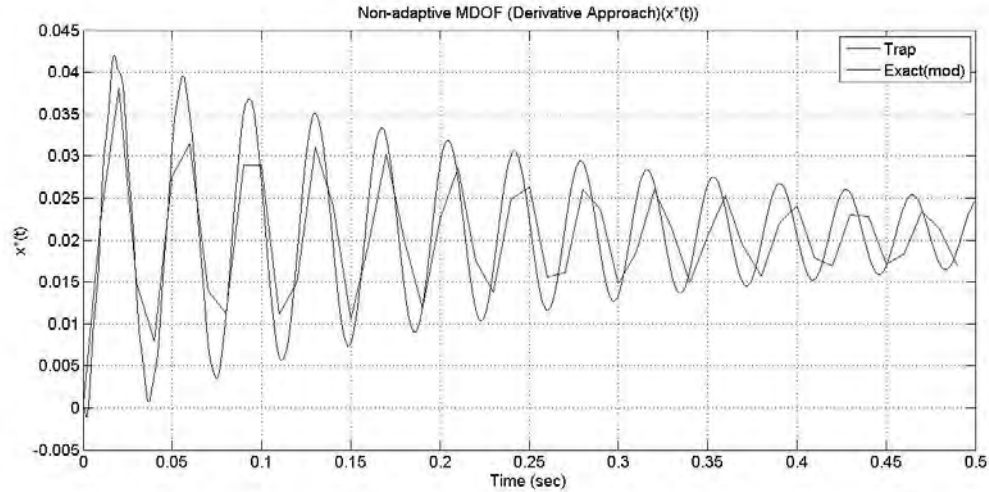


Figure 28. DOF derivative approximation 50% elemental stiffness modification and a time-step of 0.01 second.

It can be seen from Figures 27 and 28 that at a time-step of 0.01 second, the approximation is not very accurate, but it exhibits stability. It can also be seen that the 14% stiffness modification most closely follows the exact solution. Now that stability has been achieved using the derivative approach, the next question is whether the time-step can be refined to achieve accuracy.

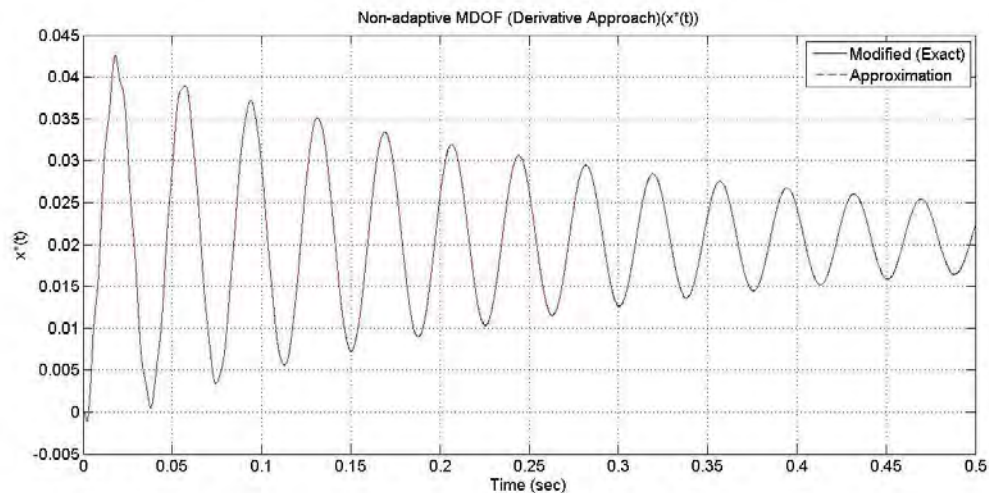


Figure 29. DOF derivative approximation 14% elemental stiffness modification and a time-step of 0.00001 second.

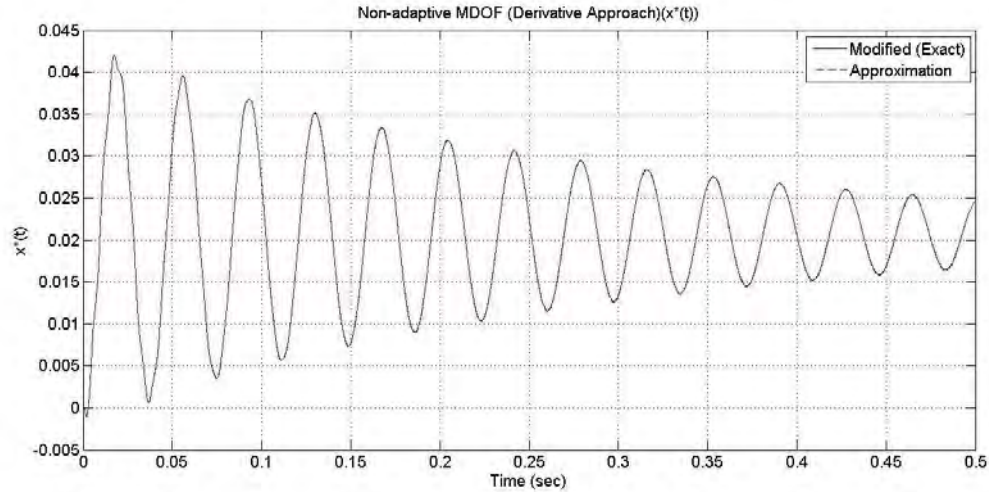


Figure 30. DOF derivative approximation 50% elemental stiffness modification and a time-step of 0.00001 second.

From Figures 29 and 30, it can be seen that not only has the derivative approximation achieved stability, but it also accurately approximates the system response. For the stiffness modification of 14%, step-size of 0.00001 second, the error is 0.1%. For the stiffness modification of 50%, step-size of 0.00001 second, the error is 0.7%. Though there are many benefits to using this approach to approximating the system response, the time needed to perform the approximation is prohibitive when the time-step is small. This is the reason Figures 29 and 30 are plotted over a small time interval.

G. THE MATRIX FORMULATION FOR THE DERIVATIVE OF THE TRANSIENT STRUCTURAL SYNTHESIS EQUATION FOR THE MDOF CANTILEVERED ALUMINUM BEAM WITH AN EXTERNALLY APPLIED STEP EXCITATION, SUBJECTED TO A STIFFNESS MODIFICATION TO AN ARBITRARY BEAM ELEMENT

The final point of the previous section was that the time to perform the approximation using the derivative approach was not desirable. This was also true for the non-derivative approach as well. Figures 25, 29, and 30, took approximately two hours to execute, and over two days on a time-scale from zero to two seconds in order to capture equilibrium of the system.

The time that it takes to approximate the MDOF systems can be attributed to the algorithms used to perform these approximations. When programming the SDOF systems in MATLAB, there are many time-saving methods that avoid using expensive time consuming loops when operating with scalar vectors. When programming these algorithms for the MDOF cases, the same time saving methods used for scalar vectors did not seem to have a direct correlation to matrix and vector arrays. As a result, these models required the use of loops to solve the approximations, and therefore more time to solve.

The matrix formulation for the derivative approach follows the formulation given in [11], and is provided in Appendix D. The elements of the resulting matrix are matrices, and the vectors of the formulation are composed of vectors. The challenge with the matrix formulation is that in order to approximate the solution on a time-step needed for sufficient accuracy, the matrix is large and requires a large amount of computer memory (RAM). In order to obtain an approximation, the time interval of approximation needs to be small. There are supercomputers that can be used with virtually unlimited memory and can quickly evaluate large systems; the goal is to develop an algorithm that can be evaluated on computers with limited memory, a common restriction.

The challenge is to determine if the matrix formulation of the derivative of the synthesis equation can produce an approximation faster than the approximation algorithms previously used. A comparison of the matrix formulation against the approximation algorithms over small time intervals was conducted and the results are given in Table 4. Table 4 gives the time it takes to construct the components of the matrix formulation, the time to solve the system, and the time it took to execute the derivative algorithm.

From Table 4 there are a few points that have to be considered prior to determining whether the matrix formulation is a viable option. First, constructing each part of the matrix formulation takes a comparable amount of time to the time it actually takes to solve the system. Constructing the matrix formulation may not have been done in an efficient manner, so the comparison should be

taken from the point at which the fully constructed matrix system is being solved. Second, on small time-intervals, the original approximation algorithms are faster than constructing and solving the matrix formulation. It is difficult to tell at this point as to whether the matrix formulation is more beneficial for time. Due to limited computer memory, constructing a matrix needed to solve the time interval that would provide a good comparison could not be accomplished. Finally, this method should not be fully discounted. The results that it produces are very accurate (0.3% error) as seen in Figure 31. If it can be shown that on a longer time scale this method is comparable to the original derivative algorithms, the matrix formulation can be a valid alternative to the derivative algorithms.

Time Interval (sec)	Construct Matrix Elements (sec)	Construct Matrix (sec)	Construct Vector (sec)	Solving Matrix System (sec)	Total time to solve (sec)	MDOF Derivative algorithm (sec)
0-0.025	20.1	50.6	0.12	9.6	83.1	20.81
0-0.050	81.3	210.0	0.24	67.8	362.3	77.3
0-0.075	183.1	477.9	0.35	223.7	883.7	170.3

Table 4. Time required to complete each part of the matrix formulation of the derivative to the integral formulation for structural synthesis.

It can be seen from Table 4 that if the entire process of constructing and solving the matrix formulation is the measure of time to be compared against the original approximation integral, then this method is not a viable alternative. It should be pointed out that the total time for the matrix formulation given in Table 4 is an average of three runs. These runs were performed to check for consistency. However, if the measure of time is the time that it takes to solve the fully constructed matrix system, then the results are inconclusive. Though the matrix formulation is faster for the first two time intervals, on the last interval it

becomes noticeably slower. Further time intervals could not be used for comparison due to the limits of computer memory.

For the time scale of 0-0.075 seconds with a time-step of 1×10^{-5} sec, the matrix size is 30000x30000, a relatively large system to be solved. Though the program was able to approximate the response for a time interval of 0 to 0.10 seconds, computer performance was severely affected and the time data gathered was too varied to establish a trend for comparison. Though the time data was too varied to be used, Figure 31 gives the result for the matrix formulation. It can be seen that this method is incredibly accurate with an error of 0.3%.

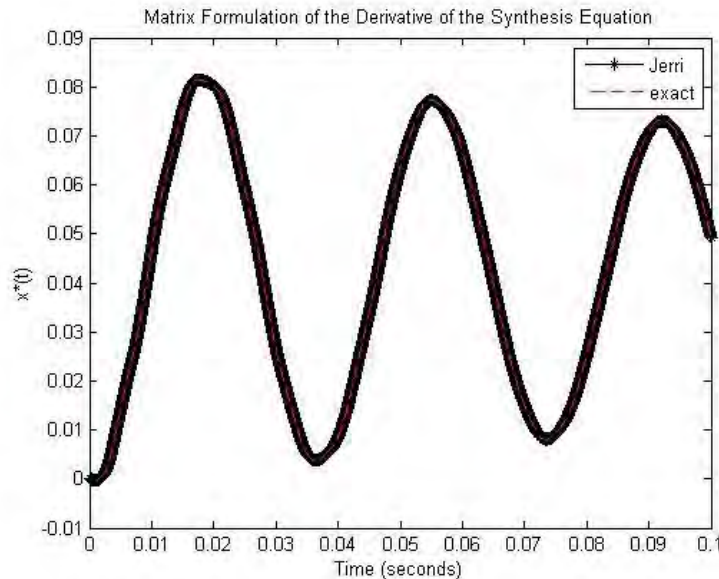


Figure 31. Matrix formulation of the derivative of the synthesis equation for the Aluminum cantilevered beam with a 100% stiffness modification and time-step of .00001 second.

THIS PAGE INTENTIONALLY LEFT BLANK

IV. CONCLUSION AND RECOMMENDATIONS

The adaptive method to solving the integral formulation for transient structural synthesis is a valid approach. As of now, the adaptive method is best used to approximate the transient response of SDOF and two DOF systems. In terms of the SDOF system, the programming methods available bypass the need to use time consuming loops allowing the approximations to be performed quickly. In terms of the two DOF system, this particular MDOF system is sufficiently simple enough that even though the use of loops is necessary, the approximations are performed with sufficient speed.

Due to computational inefficiency, the adaptive method is not an efficient alternative to solving MDOF transient structural synthesis problems over a long time scale. The next step is to study the programming methods available that can efficiently perform the adaptive algorithms using arrays of vector and matrices. Once these methods are developed, then adaptivity can be implemented.

A matrix formulation of the MDOF problem was considered to overcome the computational inefficiency due to using array operations. Unfortunately, the size of matrix needed to approximate the response of the system is very large and limited by available computer memory. Approximations can only be performed on small time intervals due to available memory, but on these small intervals, the adaptive approximation algorithms are comparable in speed. Since the analysis is assumed to be conducted by computers with limited memory, an efficient method such as a block-by-block method needs to be investigated. A conclusive comparison of the matrix formulation to the adaptive approximation algorithms cannot be made at this time.

If the structural modification is sufficiently large, then the integral formulation for transient structural analysis becomes conditionally stable. The derivative of the integral formulation for transient structural synthesis restores unconditional stability and provides velocity of the modified system response.

Keep in mind that the derivative of the synthesis equation complicates the adaptive algorithms.

Finally, once computational efficiency for MDOF problems has been established, mass and damping modifications need to be studied. Once these modifications are understood and successfully approximated, then consideration needs to be given to non-linear structural modifications.

APPENDIX A. COARSE SOLUTION FOR THE INTEGRAL EQUATION FORMULATION FOR STRUCTURAL SYNTHESIS

The MDOF integral formulation of structural synthesis is given by the following equation:

$$\{x^*(t)\} = \{x(t)\} - \int_0^t [h(t-\tau)][\Delta K]\{x^*(\tau)\} d\tau \quad (43)$$

At time-step j , the integral can be written as the sum of integrals:

$$\{x^*\}_j = \sum_{i=0}^{j-1} - \int_{t_i}^{t_{i+1}} [h(t_i-\tau)][\Delta K]\{x^*\} d\tau + \{x\}_j \quad (44)$$

Applying the trapezoidal rule:

$$\{x^*\}_j = - \sum_{i=0}^{j-1} \frac{1}{2} \delta_{i+1} ([h]_{j,i+1} [\Delta K]\{x^*\}_{i+1} + [h]_{j,i} [\Delta K]\{x^*\}_i) + \{x\}_j \quad (45)$$

$$\Rightarrow \{x^*\}_j = - \sum_{i=0}^{j-1} \frac{1}{2} \delta_{i+1} [h]_{j,i+1} [\Delta K]\{x^*\}_{i+1} - \sum_{i=0}^{j-1} \frac{1}{2} \delta_{i+1} [h]_{j,i} [\Delta K]\{x^*\}_i + \{x\}_j \quad (46)$$

$$\Rightarrow \{x^*\}_j = - \sum_{i=1}^j \frac{1}{2} \delta_i [h]_{j,i} [\Delta K]\{x^*\}_i - \sum_{i=0}^{j-1} \frac{1}{2} \delta_{i+1} [h]_{j,i} [\Delta K]\{x^*\}_i + \{x\}_j \quad (47)$$

$$\begin{aligned} \Rightarrow \{x^*\}_j &= - \sum_{i=0}^{j-1} \frac{1}{2} \delta_i [h]_{j,i} [\Delta K]\{x^*\}_i \\ &\quad - \sum_{i=0}^{j-1} \frac{1}{2} \delta_{i+1} [h]_{j,i} [\Delta K]\{x^*\}_i - \frac{1}{2} \delta_j [h]_{j,j} [\Delta K]\{x^*\}_j + \{x\}_j \end{aligned} \quad (48)$$

$$\Rightarrow \{x^*\}_j + \frac{1}{2} \delta_j [h]_{j,j} [\Delta K]\{x^*\}_j = - \sum_{i=0}^{j-1} \frac{1}{2} (\delta_i + \delta_{i+1}) [h]_{j,i} [\Delta K]\{x^*\}_i + \{x^*\}_j \quad (49)$$

$$\therefore ([I] + \frac{1}{2} \delta_j [h]_{j,j} [\Delta K]) \{x^*\}_j = - \sum_{i=0}^{j-1} \frac{1}{2} (\delta_i + \delta_{i+1}) [h]_{j,i} [\Delta K]\{x^*\}_i + \{x^*\}_j \quad (50)$$

From Equation (1), it can be seen that when $t = 0$, $\{x^*\}_0 = \{x\}_0 = \{0\}$.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX B. THE DERIVATIVE OF THE INTEGRAL EQUATION FORMULATION FOR STRUCTURAL SYNTHESIS

Starting with the MDOF integral formulation of structural synthesis:

$$\{x^*(t)\} = \{x(t)\} - \int_0^t [h(t-\tau)][\Delta K]\{x^*(\tau)\} d\tau \quad (51)$$

Taking the derivative of Equation (1) with respect to t:

$$\frac{d}{dt}\{x^*(t)\} = \frac{d}{dt}\{x(t)\} - \frac{d}{dt} \int_0^t [h(t-\tau)][\Delta K]\{x^*(\tau)\} d\tau \quad (52)$$

Recall:

$$\{x(t)\} = \int_0^t [h(t-\tau)]\{f(\tau)\} d\tau \quad (53)$$

Substituting Equation (3) into Equation (2):

$$\frac{d}{dt}\{x^*(t)\} = \frac{d}{dt} \int_0^t [h(t-\tau)]\{f(\tau)\} d\tau - \frac{d}{dt} \int_0^t [h(t-\tau)][\Delta K]\{x^*(\tau)\} d\tau \quad (54)$$

Analyzing $\frac{d}{dt}\{x(t)\}$:

$$\{\dot{x}(t)\} = \frac{d}{dt} \int_0^t [h(t-\tau)]\{f(\tau)\} d\tau \quad (55)$$

$$\begin{aligned} \frac{d}{dt} \int_0^t [h(t-\tau)]\{f(\tau)\} d\tau &= [h(t-t)]\{f(t)\} \frac{d(t)}{dt} - [h(t-0)]\{f(0)\} \frac{d(0)}{dt} \\ &\quad + \int_0^t \frac{d}{dt} ([h(t-\tau)]\{f(\tau)\}) d\tau \end{aligned} \quad (56)$$

$$\Rightarrow \frac{d}{dt} \int_0^t [h(t-\tau)]\{f(\tau)\} d\tau = 0 - 0 + \int_0^t [\dot{h}(t-\tau)]\{f(\tau)\} d\tau \quad (57)$$

$$\therefore \frac{d}{dt} \int_0^t [h(t-\tau)]\{f(\tau)\} d\tau = \{\dot{x}(t)\} = \int_0^t [\dot{h}(t-\tau)]\{f(\tau)\} d\tau \quad (58)$$

Similarly when analyzing $\frac{d}{dt} \int_0^t [h(t-\tau)][\Delta K]\{x^*(\tau)\} d\tau$:

$$\frac{d}{dt} \int_0^t [h(t-\tau)] [\Delta K] \{x^*(\tau)\} d\tau = \int_0^t [\dot{h}(t-\tau)] [\Delta K] \{x^*(\tau)\} d\tau \quad (59)$$

Substituting Equation (8) and Equation (9) into Equation (2) gives:

$$\{\dot{x}^*(t)\} = \int_0^t [\dot{h}(t-\tau)] \{f(\tau)\} d\tau - \int_0^t [\dot{h}(t-\tau)] [\Delta K] \{x^*(\tau)\} d\tau \quad (60)$$

which reduces to:

$$\{\dot{x}^*(t)\} = \{\dot{x}(t)\} - \int_0^t [\dot{h}(t-\tau)] [\Delta K] \{x^*(\tau)\} d\tau \quad (61)$$

It can be seen that when $t=0$, $\{\dot{x}^*(0)\} = \{\dot{x}(0)\} = \{0\}$. Also recall that from Equation (1), $\{x^*(0)\} = \{x(0)\} = \{0\}$ when $t=0$.

APPENDIX C. COARSE SOLUTION TO THE DERIVATIVE OF THE INTEGRAL EQUATION FORMULATION FOR STRUCTURAL SYNTHESIS

Starting with the derivative of the integral formulation for structural synthesis:

$$\{\dot{x}^*(t)\} = \{\dot{x}(t)\} - \int_0^t [\dot{h}(t-\tau)] [\Delta K] \{x^*(\tau)\} d\tau \quad (62)$$

The integral is broken into the sum of integrals;

$$\{\dot{x}^*(t)\} = - \sum_{i=0}^{j-1} \int_{t_i}^{t_{i+1}} [\dot{h}(t_i - \tau)] [\Delta K] \{x^*(\tau)\} d\tau + \{\dot{x}(t)\} \quad (63)$$

Applying the trapezoidal rule:

$$\{\dot{x}^*\}_j = - \sum_{i=0}^{j-1} \frac{1}{2} \delta_{i+1} ([\dot{h}]_{j,i+1} [\Delta K] \{x^*\}_{i+1} + [\dot{h}]_{j,i} [\Delta K] \{x^*\}_i) + \{\dot{x}\}_j \quad (64)$$

$$\{\dot{x}^*\}_j = - \sum_{i=0}^{j-1} \frac{1}{2} \delta_{i+1} ([\dot{h}]_{j,i+1} [\Delta K] \{x^*\}_{i+1}) - \sum_{i=0}^{j-1} \frac{1}{2} \delta_{i+1} ([\dot{h}]_{j,i} [\Delta K] \{x^*\}_i) + \{\dot{x}\}_j \quad (65)$$

$$\{\dot{x}^*\}_j = - \sum_{i=1}^j \frac{1}{2} \delta_i ([\dot{h}]_{j,i} [\Delta K] \{x^*\}_i) - \sum_{i=0}^{j-1} \frac{1}{2} \delta_{i+1} ([\dot{h}]_{j,i} [\Delta K] \{x^*\}_i) + \{\dot{x}\}_j \quad (66)$$

$$\begin{aligned} \{\dot{x}^*\}_j = & - \sum_{i=0}^{j-1} \frac{1}{2} \delta_i ([\dot{h}]_{j,i} [\Delta K] \{x^*\}_i) - \sum_{i=0}^{j-1} \frac{1}{2} \delta_{i+1} ([\dot{h}]_{j,i} [\Delta K] \{x^*\}_i) + \\ & - \frac{1}{2} \delta_j [\dot{h}]_{j,j} [\Delta K] \{x^*\}_j + \{\dot{x}\}_j \end{aligned} \quad (67)$$

The unknown $\{x^*\}_j$ needs to be expressed in terms of the derivative terms by means of the trapezoidal rule:

$$\{x^*\}_j = \sum_{i=0}^{j-2} \frac{1}{2} \delta_i (\{\dot{x}^*\}_i + \{\dot{x}^*\}_{i+1}) + \frac{1}{2} \delta_j (\{\dot{x}^*\}_{j-1} + \{\dot{x}^*\}_j) + \{x^*\}_0 \quad (68)$$

Substituting Equation (7) into Equation (6):

$$\begin{aligned} \{\dot{x}^*\}_j = & - \sum_{i=0}^{j-1} \frac{1}{2} (\delta_i + \delta_{i+1}) [\dot{h}]_{j,i} [\Delta K] \{x^*\}_i \\ & - \frac{1}{2} \delta_j [\dot{h}]_{j,j} [\Delta K] \left(\sum_{i=0}^{j-2} \frac{1}{2} \delta_i (\{\dot{x}^*\}_i + \{\dot{x}^*\}_{i+1}) + \frac{1}{2} \delta_j (\{\dot{x}^*\}_{j-1} + \{\dot{x}^*\}_j) + \{x^*\}_0 \right) + \{\dot{x}\}_j \end{aligned} \quad (69)$$

$$\begin{aligned}
\{\dot{x}^*\}_j &= -\sum_{i=0}^{j-1} \frac{1}{2} (\delta_i + \delta_{i+1}) [\dot{h}]_{j,i} [\Delta K] \{x^*\}_i \\
&\quad - \frac{1}{2} \delta_j [\dot{h}]_{j,j} [\Delta K] \left(\sum_{i=0}^{j-2} \frac{1}{2} \delta_i (\{\dot{x}^*\}_i + \{\dot{x}^*\}_{i+1}) \right) - \frac{1}{2} \delta_j [\dot{h}]_{j,j} [\Delta K] \left(\frac{1}{2} \delta_j \{\dot{x}^*\}_{j-1} \right) \quad (70) \\
&\quad - \frac{1}{2} \delta_j [\dot{h}]_{j,j} [\Delta K] \left(\frac{1}{2} \delta_j \{\dot{x}^*\}_j \right) - \frac{1}{2} \delta_j [\dot{h}]_{j,j} [\Delta K] \{x^*\}_0 + \{\dot{x}\}_j
\end{aligned}$$

$$\begin{aligned}
\{\dot{x}^*\}_j + \frac{1}{4} \delta_j^2 [\dot{h}]_{j,j} [\Delta K] \{\dot{x}^*\}_j &= -\sum_{i=0}^{j-1} \frac{1}{2} (\delta_i + \delta_{i+1}) [\dot{h}]_{j,i} [\Delta K] \{x^*\}_i \\
&\quad - \frac{1}{2} \delta_j [\dot{h}]_{j,j} [\Delta K] \left(\sum_{i=0}^{j-2} \frac{1}{2} \delta_i (\{\dot{x}^*\}_i + \{\dot{x}^*\}_{i+1}) \right) \quad (71) \\
&\quad - \frac{1}{4} \delta_j^2 [\dot{h}]_{j,j} [\Delta K] \{\dot{x}^*\}_{j-1} - \frac{1}{2} \delta_j [\dot{h}]_{j,j} [\Delta K] \{x^*\}_0 + \{\dot{x}\}_j
\end{aligned}$$

$$\begin{aligned}
\therefore ([I] + \frac{1}{4} \delta_j^2 [\dot{h}]_{j,j} [\Delta K] \{\dot{x}^*\}_j) \{\dot{x}^*\}_j &= -\sum_{i=0}^{j-1} \frac{1}{2} (\delta_i + \delta_{i+1}) [\dot{h}]_{j,i} [\Delta K] \{x^*\}_i \\
&\quad - \frac{1}{2} \delta_j [\dot{h}]_{j,j} [\Delta K] \left(\sum_{i=0}^{j-2} \frac{1}{2} \delta_i (\{\dot{x}^*\}_i + \{\dot{x}^*\}_{i+1}) \right) \quad (72) \\
&\quad - \frac{1}{4} \delta_j^2 [\dot{h}]_{j,j} [\Delta K] \{\dot{x}^*\}_{j-1} - \frac{1}{2} \delta_j [\dot{h}]_{j,j} [\Delta K] \{x^*\}_0 + \{\dot{x}\}_j
\end{aligned}$$

APPENDIX D. THE MATRIX FORMULATION TO THE DERIVATIVE OF THE TRANSIENT STRUCTURAL SYNTHESIS EQUATION

In order to derive the matrix formulation, it is helpful to look at a number of iterations. The derivative of the transient structural synthesis equation takes the form:

$$\{\dot{x}^*\} = \{\dot{x}\} - \int_0^t [\dot{h}(t-\tau)] [\Delta K] \{x^*(\tau)\} d\tau \quad (73)$$

Let $t=0$:

$$\{\dot{x}^*(0)\} = \{\dot{x}(0)\} - \int_0^0 [\dot{h}(0)] [\Delta K] \{x^*(\tau)\} d\tau \quad (74)$$

$$\therefore \{\dot{x}^*(0)\} = \{\dot{x}(0)\} \quad (75)$$

Let $t=1$:

$$\{\dot{x}^*(1)\} = \{x^*(1)\} - \int_0^1 [\dot{h}(1-\tau)] [\Delta K] \{x^*(\tau)\} d\tau \quad (76)$$

$$\{\dot{x}^*(1)\} = \{\dot{x}(1)\} - \Delta t \left(\frac{1}{2} [\dot{h}]_{10} [\Delta K] \{x^*(0)\} + \frac{1}{2} [\dot{h}]_{11} [\Delta K] \{x^*(1)\} \right) \quad (77)$$

$$\{\dot{x}^*(1)\} = \{\dot{x}(1)\} - \frac{\Delta t}{2} [\dot{h}]_{10} [\Delta K] \{x^*(0)\} - \frac{\Delta t}{2} [\dot{h}]_{11} [\Delta K] \left(\frac{\Delta t}{2} \{\dot{x}^*(0)\} + \frac{\Delta t}{2} \{\dot{x}^*(1)\} + \{x^*(0)\} \right) \quad (78)$$

$$\begin{aligned} \{\dot{x}(1)\} &= \left(\frac{\Delta t}{2} [\dot{h}]_{10} [\Delta K] + \frac{\Delta t}{2} [\dot{h}]_{11} [\Delta K] + \right) \{x^*(0)\} \\ &\quad + \frac{\Delta t^2}{4} [\dot{h}]_{11} [\Delta K] \{\dot{x}^*(0)\} \\ &\quad \left(I + \frac{\Delta t^2}{4} [\dot{h}]_{11} [\Delta K] \right) \{\dot{x}^*(1)\} \end{aligned} \quad (79)$$

Following the same process for $t=2$, and $t=3$ gives:

$$\begin{aligned}
\{\dot{x}(2)\} = & \left(\frac{\Delta t}{2} [\dot{h}]_{20} [\Delta K] + \Delta t [\dot{h}]_{21} [\Delta K] + \frac{\Delta t}{2} [\dot{h}]_{22} [\Delta K] + \right) \{x^*(0)\} \\
& + \left(\frac{\Delta t^2}{2} [\dot{h}]_{21} [\Delta K] + \frac{\Delta t^2}{4} [\dot{h}]_{22} [\Delta K] \right) \{\dot{x}^*(0)\} \\
& + \left(\frac{\Delta t^2}{2} [\dot{h}]_{21} [\Delta K] + \frac{\Delta t^2}{2} [\dot{h}]_{22} [\Delta K] \right) \{\dot{x}^*(1)\} \\
& + \left(I + \frac{\Delta t^2}{4} [\dot{h}]_{22} [\Delta K] \right) \{\dot{x}^*(2)\}
\end{aligned} \tag{80}$$

$$\begin{aligned}
\{\dot{x}(3)\} = & \left(\frac{\Delta t}{2} [\dot{h}]_{30} [\Delta K] + \Delta t [\dot{h}]_{31} [\Delta K] + \Delta t [\dot{h}]_{32} [\Delta K] + \frac{\Delta t}{2} [\dot{h}]_{33} [\Delta K] + \right) \{x^*(0)\} \\
& + \left(\frac{\Delta t^2}{2} [\dot{h}]_{31} [\Delta K] + \frac{\Delta t^2}{2} [\dot{h}]_{32} [\Delta K] + \frac{\Delta t^2}{4} [\dot{h}]_{33} [\Delta K] \right) \{\dot{x}^*(0)\} \\
& + \left(\frac{\Delta t^2}{2} [\dot{h}]_{31} [\Delta K] + \Delta t^2 [\dot{h}]_{32} [\Delta K] + \frac{\Delta t^2}{2} [\dot{h}]_{33} [\Delta K] \right) \{\dot{x}^*(1)\} \\
& + \left(\frac{\Delta t^2}{2} [\dot{h}]_{32} [\Delta K] + \frac{\Delta t^2}{2} [\dot{h}]_{33} [\Delta K] \right) \{\dot{x}^*(2)\} \\
& + \left(I + \frac{\Delta t^2}{4} [\dot{h}]_{33} [\Delta K] \right) \{\dot{x}^*(3)\}
\end{aligned} \tag{81}$$

Let $t=j$:

$$\begin{aligned}
\{\dot{x}(j)\} = & \left(\frac{\Delta t}{2} [\dot{h}]_{j0} [\Delta K] + \Delta t [\dot{h}]_{j1} [\Delta K] + \dots + \frac{\Delta t}{2} [\dot{h}]_{jj} [\Delta K] + \right) \{x^*(0)\} \\
& + \left(\frac{\Delta t^2}{2} [\dot{h}]_{j1} [\Delta K] + \frac{\Delta t^2}{2} [\dot{h}]_{j2} [\Delta K] + \dots + \frac{\Delta t^2}{4} [\dot{h}]_{jj} [\Delta K] \right) \{\dot{x}^*(0)\} \\
& + \left(\frac{\Delta t^2}{2} [\dot{h}]_{j1} [\Delta K] + \Delta t^2 [\dot{h}]_{j2} [\Delta K] + \dots + \frac{\Delta t^2}{2} [\dot{h}]_{jj} [\Delta K] \right) \{\dot{x}^*(1)\} \\
& \vdots \\
& + \left(I + \frac{\Delta t^2}{4} [\dot{h}]_{jj} [\Delta K] \right) \{\dot{x}^*(j)\}
\end{aligned} \tag{82}$$

Assuming rest initial conditions:

$$\{x(0)\} = \{\dot{x}(0)\} = \{x^*(0)\} = \{\dot{x}^*(0)\} = \{0\}$$

Putting the system of equations into matrix form:

$$[A] = \begin{bmatrix} \left(I + \frac{\Delta t^2}{4} [\ddot{h}]_{11} [\Delta K] \right) & [0] & [0] & [0] \\ \left(\frac{\Delta t^2}{2} [\ddot{h}]_{21} [\Delta K] + \frac{\Delta t^2}{2} [\ddot{h}]_{22} [\Delta K] \right) & \left(I + \frac{\Delta t^2}{4} [\ddot{h}]_{11} [\Delta K] \right) & [0] & [0] \\ \vdots & \ddots & \dots & [0] \\ \left(\frac{\Delta t^2}{2} [\ddot{h}]_{j1} [\Delta K] + \Delta t^2 [\ddot{h}]_{j1} [\Delta K] + \dots + \frac{\Delta t^2}{2} [\ddot{h}]_{j22} [\Delta K] \right) & \dots & \left(\frac{\Delta t^2}{2} [\ddot{h}]_{jj-1} [\Delta K] + \frac{\Delta t^2}{2} [\ddot{h}]_{jj} [\Delta K] \right) & \left(I + \frac{\Delta t^2}{4} [\ddot{h}]_{11} [\Delta K] \right) \end{bmatrix}$$

$$\{\dot{x}^*\} = \begin{Bmatrix} \{\dot{x}^*(1)\} \\ \{\dot{x}^*(2)\} \\ \vdots \\ \{\dot{x}^*(j)\} \end{Bmatrix}$$

$$\{\dot{x}\} = \begin{Bmatrix} \{\dot{x}(1)\} \\ \{\dot{x}(2)\} \\ \vdots \\ \{\dot{x}(3)\} \end{Bmatrix}$$

$$\therefore [A] \{\dot{x}^*\} = \{\dot{x}\} \quad (83)$$

Equation 83 is the equation to be solved.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX E. MATLAB CODE FOR A SDOF MASS-SPRING SYSTEM WITH EXTERNALLY APPLIED STEP FUNCTION SUBJECTED TO A STIFFNESS MODIFICATION (SUPPORTING FUNCTIONS INCLUDED)

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%This is the main program for an adaptive SDOF mass-spring problem %
%subjected to a periodic forcing function. %
% Created by Keenan L. Coleman 8/25/2014 %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
close all;
clear all;
clc;
% Initializing the SDOF parameter for the exact solution:
damp_ratio = 0.1; % damping ratio
k = 100; % spring constant (N/m)
delk = .1*k; % stiffness modification
mass = 0.1; % mass (kg)
omega2 = sqrt((k + delk)/mass); % natural frequency due to
stiffness modification (used to calculate the exact solution)
dampfreq2 = omega2*sqrt(1-damp_ratio^2); % damped frequency due to
stiffness modification (used to calculation the exact solution)
% Initializing the adaptive algorithm variables:
% Time variables:
time_start = 0; % start time (sec)
time_end = 2; % end time (sec)
delt = .01; % initial time-step/time difference (sec)
min_del = 0.00001; % minimum time step that terminates the adaptive
process during step halving
t = time_start:min_del:time_end; % time interval for the function
handle to the exact solution to the modified response
t_step = [0 delt]; % initializing the vector of time-steps
t_diff = [delt 0]; % initializing the vector of time differences
between time-steps for the kernel
t_stepcount = time_start + delt; % the value of the current time-
step
h = [0 (delt - time_start)]; % initializing the vector of time
differences for the
h2 = [0 (delt - time_start)];
% Spacial variables:
x_initial = 0;
x_coarse = [x_initial];
x_half = [x_initial];
x_fine = [x_initial];
x_approx = [x_initial];
count = 1;
count2 = 0;
double_count = 0;
half_count = 0;
max_error = .00001; % maximum error between fine and coarse solution
to determine step-halving or step-doubling
% The exact solution...

```

```

exact_solution = @(t) (1/(mass*dampfreq2))*exp(-
damp_ratio*omega2*t).*(dampfreq2*exp(damp_ratio*omega2*t)-
damp_ratio*omega2*sin(dampfreq2*t)-
dampfreq2*cos(dampfreq2*t))/(damp_ratio^2*omega2^2+dampfreq2^2);
exact_solution = exact_solution(t);
tic
% Solving for x(t), the unmodified solution (uses the Trap rule
solution).
[x_solution] = x_solve_2(time_start, time_end,delt, min_del);
while t_stepcount<=time_end
    % Calling values for x(t) ( This is g(t) in Dr. Neta's paper).
    [x_result,x_result_half] = x_solution_call_2(time_start,
min_del,time_end,t_step(count) ,t_step(count+1),x_solution);
    % This is the kernel function
    [ kern_elem,kern1,kernhalf ] = G2_kernel( t_diff,t_step,count,delt
);
    % Course Solution
    [x_coarse, x_coarsevalue] =
G_course_solution_2(h,x_coarse,x_result,kern_elem,kern1,count);
    % x-half solution
    [x_half, x_halfvalue] =
G_half_solution_2(h,x_coarse,x_half,kernhalf,kern1,count,x_result_half)
;
    % Fine Solution
    [x_fine, x_finevalue] =
G_fine_solution_2(h,x_coarse,x_fine,x_halfvalue,x_result,kern_elem,kern
1,kernhalf,count);
    % evaluating the error
    x_value = [x_finevalue,x_coarsevalue];
    relerror = abs(diff(x_value))/abs(max(x_coarse));
    relerror = 0.99*relerror;
    int1 = find(t==t_stepcount);
    t_stepcount;
    exact = exact_solution(int1);
    x_finevalue;
    x_coarsevalue;
    relerror;
    % adjusting the time-step
    if delt <= min_del
        sprintf('Minimum time difference exceeded')
        break
    end
    if relerror < max_error
    x_approx(count+1) = x_finevalue;
    x_coarse(count+1) = x_finevalue;
    if t_stepcount == time_end
        break
    end
    count = count +1;
    count2 = count2 + 1;
    double_count = double_count + 1;
    delt = 2*delt;
    h(count+1) = delt;
    h2(count2+1) = delt;
    t_step(count+1) = t_step(count) + delt;

```

```

t_stepcount = t_step(count+1);
t_stepvalue = t_step(count) + delt;
[r1,c1] = size(t_step);
    for ii = 1:c1
        t_diff(ii) = t_step(end) - t_step(ii);
    end
end
if relerror > max_error
    delt = delt/2;
    count2 = count2 + 1;
    half_count = half_count + 1;
    h(count+1) = delt;
    h2(count2+1) = delt;
    t_step(count+1) = t_step(count+1) - delt;
    t_stepcount = t_step(count+1);
    t_stepvalue = t_step(count+1);
    [r1,c1] = size(t_step);
    for ii = 1:c1
        t_diff(ii) = t_step(end) - t_step(ii);
    end
    x_coarse = x_coarse(1:count);
    x_half = x_half(1:count);
    x_fine = x_fine(1:count);
end
if t_stepcount == time_end
    break
end

if t_stepvalue > time_end
    t_step(count+1) = time_end;
    h(count +1) = time_end - t_step(count);
    h2(count2 +1) = time_end - t_step(count);
    t_stepcount = t_step(count +1);
    delt = h(count+1);
    [r1,c1] = size(t_step);
    for ii = 1:c1
        t_diff(ii) = t_step(end) - t_step(ii);
    end
    x_coarse = x_coarse(1:count);
    x_half = x_half(1:count);
    x_fine = x_fine(1:count);
end
end
toc
double_count
half_count
total_count = count2
% Plotting the results:
% Change default axes fonts.
set(0,'DefaultAxesFontName','Arial')
set(0,'DefaultAxesFontSize', 14)
% Change default text fonts.
set(0,'DefaultTextFontname','Arial')
set(0,'DefaultTextFontSize', 14)
figure(1)

```

```

set(gcf,'WindowStyle','Docked')
plot(t_step,x_approx,'kd',t,exact_solution,'k')
legend('x*(t)','Exact')
xlabel('Time (sec)')
ylabel('x*(t) (modified displacement)')
title('x*(t) vs. Time')
grid on
figure(2)
set(gcf,'WindowStyle','Docked')
[r1,c1] = size(h2);
tt = 1:c1;
plot(tt,h2,'bd')
xlabel('Step Number')
ylabel('Step Size')
title('Step Size vs. Step Number')
grid on
axis([0,total_count,0,.045])
figure(3)
set(gcf,'WindowStyle','Docked')
[r1,c1] = size(h2);
tt = 1:c1;
plot(tt,h2,'bd')
xlabel('Step Number')
ylabel('Step Size')
title('Step Size vs. Step Number')
grid on
axis([0,150,0,.045])
% Evaluating the error...
exact_solution_error = @(t_step) (1/(mass*dampfreq2))*exp(-
damp_ratio*omega2*t_step).*(dampfreq2*exp(damp_ratio*omega2*t_step)-
damp_ratio*omega2*sin(dampfreq2*t_step)-
dampfreq2.*cos(dampfreq2*t_step))/(damp_ratio^2*omega2^2+dampfreq2^2);
exact_solution_error = exact_solution_error(t_step);
max_diff = max(abs(exact_solution_error - x_approx));
min_diff = min(abs(exact_solution_error - x_approx));
error_eval = max(abs(exact_solution_error -
x_approx))/max(exact_solution_error)
error_eval2 = min(abs(exact_solution_error -
x_approx))/max(exact_solution_error)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%This is the coarse solution algorithm as derived by Gordis and Neta.%
%Created by Keenan Coleman on 8/25/2014 %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [x_coarse,x_coarsevalue] =
G_course_solution_2(h,x_coarse,x_result,kern_elem,kern1,count)

    lhsCoeff_coarse = 1 - 0.5*h(end)*kern1(end);
    for ss = 1:count
        h_term(ss) = (h(ss+1) + h(ss));
    end
    x_product = 0.5*h_term.*kern_elem.*x_coarse;
    x_sum = sum(x_product) + x_result;
    x_coarse(count+1) = x_sum/lhsCoeff_coarse;
    x_coarsevalue = x_sum/lhsCoeff_coarse;

```

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%This is the half solution algorithm as derived by Gordis and Neta%
%Created by Keenan Coleman 8/25/2014                                     %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [x_half,x_halfvalue] =
G_half_solution_2(h,x_coarse,x_half,kernhalf,kern1,count,x_result_half)
    lhsCoeff_half = 1 - 0.25*h(end)*kern1(end);
    x_product_half = 0;

    if count >= 2;
        for ss = 1:count-1
            h_term(ss) = (h(ss+1) + h(ss));
        end
        x_product_half = 0.5*h_term.*kernhalf(1:count -
1).*x_coarse(1:count -1);
    end
    x_sum = sum(x_product_half) + 0.5*(h(count) + 0.5*h(count +
1))*kernhalf(count)*x_coarse(count) + x_result_half;
    x_half(count+1) = x_sum/lhsCoeff_half;
    x_halfvalue = x_sum/lhsCoeff_half;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%This is the half solution algorithm as derived by Gordis and Neta%
%Created by Keenan Coleman 8/25/2014                                     %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [x_half,x_halfvalue] =
G_half_solution_2(h,x_coarse,x_half,kernhalf,kern1,count,x_result_half)
    lhsCoeff_half = 1 - 0.25*h(end)*kern1(end);
    x_product_half = 0;

    if count >= 2;
        for ss = 1:count-1
            h_term(ss) = (h(ss+1) + h(ss));
        end
        x_product_half = 0.5*h_term.*kernhalf(1:count -
1).*x_coarse(1:count -1);
    endd
    x_sum = sum(x_product_half) + 0.5*(h(count) + 0.5*h(count +
1))*kernhalf(count)*x_coarse(count) + x_result_half;
    x_half(count+1) = x_sum/lhsCoeff_half;
    x_halfvalue = x_sum/lhsCoeff_half;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% This calculates the kernel at the time-steps and half time-steps%
%as required by the algorithms derived by Gordis and Neta.             %
%Created by Keenan Coleman on 8/25/2014                                 %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [ kern_elem,kern1,kernhalf ] = G2_kernel(
t_diff,t_step,count,delt )

% intializing the SDOF parameters...

```

```

damp_ratio = 0.1;
k = 100;
delk = .1*k;
mass = 0.1;
omega1 = sqrt(k/mass);
a = damp_ratio*omega1;
dampfreq = omega1*sqrt( 1 - damp_ratio^2);
[row1,column1] = size (t_diff);
for ii = 1:(column1-1)
    t_half(ii) = (t_step(count)+t_step(count+1))/2 - t_step(ii);
end
kern = @(t_diff) (-delk / (mass*dampfreq))*exp(-
a*t_diff).*sin(dampfreq*t_diff);
kern1 = kern(t_diff);
kernhalf = kern(t_half);
kern_elem = kern1(1:end-1);
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% This function calls the value for the pre-calculated baseline %
%response. %
%Created by Keenan Coleman on 8/25/2014 %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [x_result,x_result_half] = x_solution_call_2(time_start,
min_del,time_end, t_step1, t_step2,x_solution)

xx = time_start:min_del:time_end;
c1 = find(xx >= t_step2);
c2 = find(xx >= (t_step1 +t_step2)/2);
c1 = min(c1);
c2 = min(c2);
x_result = x_solution(c1);
x_result_half = x_solution(c2);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%This function calculates the baseline response whose%
%values will be called as needed. %
%Created by Keenan Coleman on 8/25/2014 %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [x_solution] = x_solve_2(time_start, time_end,delt, min_del)

damp_ratio = 0.1;
k = 100;
delk = .1*k;
mass = 0.1;
omega1 = sqrt(k/mass);
a = damp_ratio*omega1;
dampfreq = omega1*sqrt( 1 - damp_ratio^2);
count1 = 1;
count2 = 2;
g1(1) = 0;
t = time_start:delt:time_end;
% Implementing the algorithm:
for jj = time_start + delt:delt:time_end
for ii = 1:count2

```

```

        f1(ii) = 1/(mass*dampfreq)*exp(-a*(jj-t(ii)))*sin(dampfreq*(jj-
t(ii)));
    end
    f1(1) = .5*f1(1);
    f1(end) = .5*f1(end);
    g_value = delt*sum(f1);
    g1(count1+1) = g_value;
    count1 = count1 + 1;
    count2 = count2 + 1;
end
xx = time_start:min_del:time_end;
yy = spline(t,g1,xx);
x_solution = yy;
exactunmod = 1/(mass*dampfreq)*exp(-a*t).*(dampfreq*exp(a*t)-
a*sin(dampfreq*t)-dampfreq*cos(dampfreq*t))/(a^2 + dampfreq^2);

```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX F. MATLAB CODE FOR A SDOF MASS-SPRING SYSTEM WITH EXTERNALLY APPLIED PERIODIC FUNCTION SUBJECTED TO A STIFFNESS MODIFICATION (SUPPORTING FUNCTIONS INCLUDED).

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% This models a SDOF problem with a periodic forcing function.%
% Created by Keenan Coleman on 8/25/2014
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
close all;
clear all;
clc;
% Initializing variables:
delt = .01;           % Initial step size
t_start = 0;          % Start time
t_stop = 2;           % Stop time
time = t_start : .00001 : t_stop ;% Timeline
nStep = length(time); % Number of steps
min_step = .00001;    % Minimum step size (necessary for adaptive
method)
% Physical Parameters:
m = 0.1;              % Mass
k = 100;              % Spring Stiffness
wn = sqrt(k/m);       % Natural frequency in rads/sec
fn = wn/(2*pi);       % Natural frequency in Hz
z = 0.01;             % Damping ratio
wd = wn * sqrt(1 - z^2); % Damped frequency
delk = 1 * k;         % Change in stiffness

% Evaluating the baseline response:
tic
[ x_base ] = x_baseline(t_start, t_stop, min_step, m, wn, wd, z);
% Implementing the adaptive algorithms from the Neta paper:
% Initializing the algorithm variables:
max_error = .0001;
t_step = [0 delt];
t_diff = [delt 0];
x_initial = 0;
h = [0 (delt - t_start)];
h2 = [0 (delt - t_start)];
x_coarse = [x_initial];
x_half = [x_initial];
x_fine = [x_initial];
x_approx = [x_initial];
count = 1;
t_stepcount = t_start + delt;
count = 1;
count2 = 0;
double_count = 0;
half_count = 0;

```

```

tic
while t_stepcount<=t_stop
    % Calling the x_base and half x_base values needed for the
    coarse,half
    % and fine values.
    [base_result,base_result_half] = x_base_call(t_start,
min_step,t_stop,t_step(count) ,t_step(count+1),x_base);
    % Evaluating the necessary kernel values:
    [ kern_elem,kern1,kernhalf ] = kernel_eval( m,wn,wd,z,
t_diff,t_step,count );
    % Evaluating the coarse solution:
    [x_coarse,x_coarsevalue] =
x_coarse_solution(h,x_coarse,base_result,kern_elem,kern1,count,delk);
    % Evaluating the half-solution:
    [x_half,x_halfvalue] =
x_half_solution(h,x_coarse,x_half,kernhalf,kern1,count,base_result_half
,delk);
    % Evaluating the fine solution:
    [x_fine, x_finevalue] =
x_fine_solution(h,x_coarse,x_fine,x_halfvalue,base_result,kern_elem,ker
n1,kernhalf,count,delk);
    % evaluating the error
    if x_coarse==0
        relerror = 0;
    else
        x_value = [x_finevalue,x_coarsevalue];
        relerror = abs(diff(x_value))/abs(max(x_coarse));
        relerror = 0.99*relerror;
    end
    % adjusting the time-step
    if delt <= min_step
        sprintf('Minimum time difference exceeded')
        break
    end
    if relerror < max_error
        x_approx(count+1) = x_finevalue;
        if t_stepcount == t_stop
            break
        end
        count = count +1;
        count2 = count2 + 1;
        double_count = double_count + 1;
        delt = 2*delt;
        h(count+1) = delt;
        h2(count2+1) = delt;
        t_step(count+1) = t_step(count) + delt;
        t_stepcount = t_step(count+1);
        t_stepvalue = t_step(count) + delt;
        [r1,c1] = size(t_step);
        for ii = 1:c1
            t_diff(ii) = t_step(end) - t_step(ii);
        end
    end
end

```

```

    if relerror > max_error
        delt = delt/2;
        count2 = count2 + 1;
        half_count = half_count + 1;
        h(count+1) = delt;
        h2(count2+1) = delt;
        t_step(count+1) = t_step(count+1) - delt;
        t_stepcount = t_step(count+1);
        t_stepvalue = t_step(count+1);
        [r1,c1] = size(t_step);
        for ii = 1:c1
            t_diff(ii) = t_step(end) - t_step(ii);
        end
        x_coarse = x_coarse(1:count);
        x_half = x_half(1:count);
        x_fine = x_fine(1:count);
    end

    if t_stepcount == t_stop
        break
    end
    if t_stepvalue > t_stop
        t_step(count+1) = t_stop;
        h(count +1) = t_stop - t_step(count);
        h2(count2 +1) = t_stop - t_step(count);
        t_stepcount = t_step(count +1);
        t_stepcount = t_step(count +1);
        delt = h(count+1);
        [r1,c1] = size(t_step);
        for ii = 1:c1
            t_diff(ii) = t_step(end) - t_step(ii);
        end
        x_coarse = x_coarse(1:count);
        x_half = x_half(1:count);
        x_fine = x_fine(1:count);
    end
end
toc
double_count
half_count
total_count = count2
% The Exact Solution:
delt2 = .00001;
k2 = 100 + delk;
wnChk = sqrt(k2/m);
fn = wnChk/(2*pi);
wd = wnChk * sqrt(1 - z^2);
omega = 4;
W = 2 * pi * omega;
fAmp = 1.0;
f = fAmp*sin(W*time);
% f = ones(size(time));
hchk = fModalIRFkernel(k2,m,z,time);
xchk = conv(hchk,f);

```

```

xchk = xchk(1:length(hchk)) * delt2;
% Plotting the results:
% Change default axes fonts.
set(0,'DefaultAxesFontName','Arial')
set(0,'DefaultAxesFontSize',14)
% Change default text fonts.
set(0,'DefaultTextFontname','Arial')
set(0,'DefaultTextFontSize',14)
figure(1)
set(gcf,'WindowStyle','Docked')
plot(t_step(1:end),x_approx,'dk',time,xchk)
legend('x*(t)','Exact')
xlabel('Time (sec)')
ylabel('x*(t)')
title('x*(t) vs. Time')
grid on
figure(2)
set(gcf,'WindowStyle','Docked')
[r1,c1] = size(h2);
tt = 1:c1;
plot(tt,h2,'kd')
xlabel('Step Number')
ylabel('Step Size')
title('Step Size vs. Step Number')
grid on
figure(3)
set(gcf,'WindowStyle','Docked')
[r1,c1] = size(h2);
tt = 1:c1;
plot(tt,h2,'kd')
xlabel('Step Number')
ylabel('Step Size')
title('Step Size vs. Step Number')
grid on
axis([0,250,0,.025])
% Determining the error:
[row1,col1] = size(t_step);
for rr = 1:col1
    xact_find = find(time>=t_step(rr));
    xact_min = min(xact_find);
    xact_sol(rr) = xchk(xact_min);
end
max_diff = max(abs(xact_sol - x_approx));
error_eval = max(abs(xact_sol - x_approx))/max(abs(xact_sol))
error_eval2 = min(abs(xact_sol - x_approx))/max(abs(xact_sol))

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%This function is the coarse solution as derived by Gordis and Neta%
%Created by Keenan Coleman on 8/25/14 %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [x_coarse,x_coarsevalue] =
x_course_solution(h,x_coarse,base_result,kern_elem,kern1,count,delk)

    lhsCoeff_coarse = 1 + 0.5*delk*h(end)*kern1(end);
    for ss = 1:count

```

```

        h_term(ss) = (h(ss+1) + h(ss));
    end
    x_product = 0.5*delk*h_term.*kern_elem.*x_coarse;
    x_sum = base_result - sum(x_product);
    x_coarse(count+1) = x_sum/lhsCoeff_coarse;
    x_coarsevalue = x_sum/lhsCoeff_coarse;
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%This function is the half solution as derived by Gordis and Neta%
%Created by Keenan Coleman on 8/25/14                                     %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [x_half,x_halfvalue] =
x_half_solution(h,x_coarse,x_half,kernhalf,kern1,count,base_result_half
,delk)

    lhsCoeff_half = 1 + 0.25*delk*h(end)*kern1(end);
    x_product_half = 0;

    if count >= 2;
        for ss = 1:count-1
            h_term(ss) = (h(ss+1) + h(ss));
        end
        x_product_half = 0.5*delk*h_term.*kernhalf(1:count -
1).*x_coarse(1:count -1);
    end
    x_sum = -sum(x_product_half) - 0.5*delk*(h(count) + 0.5*h(count +
1))*kernhalf(count)*x_coarse(count) + base_result_half;
    x_half(count+1) = x_sum/lhsCoeff_half;
    x_halfvalue = x_sum/lhsCoeff_half;
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%This is the half solution as derived by Gordis and Neta%
%Created by Keenan Coleman on 8/25/14                                     %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [x_fine, x_finevalue] =
x_fine_solution(h,x_coarse,x_fine,x_halfvalue,base_result,kern_elem,ker
n1,kernhalf,count,delk)

    lhsCoeff_fine = 1 - 0.25*h(end)*kern1(end);
    x_product = 0;
    if count >= 2;
        for ss = 1:count-1
            h_term(ss) = (h(ss+1) + h(ss));
        end
        x_product = 0.5*delk*h_term.*kern1(1:count -1).*x_coarse(1:count
-1);
    end
    x_sum = -sum(x_product) - 0.25*delk*(h(count+1) +
2*h(count))*kern_elem(count)*x_coarse(count) -
0.5*delk*h(end)*kernhalf(count)*x_halfvalue + base_result;
    x_fine(count+1) = x_sum/lhsCoeff_fine;
    x_finevalue = x_sum/lhsCoeff_fine;

```

```

end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%This function calculates the baseline response.%
%Created by Keenan Coleman on 8/25/14          %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [ x_base ] = x_baseline(t_start, t_stop, min_step, m, wn, wd,
z)

% Initializing the variables:
t = t_start:min_step:t_stop;
% The kernel function:
kern_value = exp(-z*wn*(t)) .* sin(wd * (t)) / (m*wd);
% Periodic forcing function parameters:
omega = 4;
W = 2 * pi * omega;
fAmp = 1.0;
% The forcing function (this is the only place the forcing function is
needed):
[force_value] = fAmp*sin(W*t);
% force_value = ones(size(t));
% The convolution result:
x_base = conv(kern_value, force_value);
x_base = x_base(1:length(kern_value)) * min_step;
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%This function calls the given base resonse for the given time-step%
%Created by Keenan Coleman on 8/25/14          %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [base_result,base_result_half] = x_base_call(time_start,
min_del,time_stop, t_step1,t_step2,x_base)

xx = time_start:min_del:time_stop;
c1 = find(xx >= t_step2);
c2 = find(xx >= (t_step1 + t_step2)/2);
c1 = min(c1);
c2 = min(c2);
base_result = x_base(c1);
base_result_half = x_base(c2);
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%This function evaluates the kernel at each time-step and half%
%time-step needed for the algorithms derived by Gordis and %
%Neta. %
%Created by Keenan Coleman on 8/25/14          %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [ kern_elem,kern1,kernhalf ] =
kernel_eval(m,wn,wd,z,t_diff,t_step,count)

[row1,column1] = size (t_diff);
for ii = 1:(column1-1)
    t_half(ii) = (t_step(count)+t_step(count+1))/2 - t_step(ii);

```

```

end
kern = @(t_diff) (1/(m*wd))*exp(-z*wn*t_diff).*sin(wd*t_diff);
kern1 = kern(t_diff);
kernhalf = kern(t_half);
kern_elem = kern1(1:end-1);
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%This function evaluates the IRF for values of t%
%Created by Keenan Coleman on 8/25/14      %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function ModeIRF = fModalIRFkernel(k,m,z,t,tau)

wn = sqrt(k/m);
wd = wn * sqrt(1 - z^2);
if nargin == 4
    ModeIRF = exp(-z*wn*(t)) .* sin(wd * (t)) / (m*wd);
elseif nargin == 5
    ModeIRF = exp(-z*wn*(t-tau)) .* sin(wd * (t-tau)) / (m*wd);
End

```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX G. MATLAB CODE FOR THE FINITE ELEMENT MODEL OF THE ALUMINUM CANTILEVERED BEAM

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%This is the finite-element beam code that gives the exact solution%
%used to compare the results of all of the MDOF models.           %
%Created by Keenan Coleman on 8/25/2014                          %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

close all;
clear all;
clc;
format long;

% Problem givens
modulus = 10e6;
density = 2.5879e-4 ;
beamLength = 120;
beamHeight = 12;
beamWidth = 4;
numelem = 5;
dampingratio = .02;
rootdamp = (1-dampingratio^2)^(.5);
area = beamHeight*beamWidth;
moment = beamWidth*beamHeight^3/12;
l = beamLength/numelem;
alpha = density * area;
numdof = numelem*2+2;
precisefreq =
3.52/(2*pi)*sqrt(modulus*moment/(beamLength^4*area*density));
t = 0:.0005:.5;

% This applies the necessary applied loads and moments.
forceVec = zeros(numdof,1);
count = 1;
query = 'y';
while query == 'y' & count <= numdof
    nodeNum = 11;
    while nodeNum > numdof
        fprintf('The input exceeds vector dimensions\n');
        nodeNum = input('Enter the applicable DOF: ');
    end
    forceVec(nodeNum) = 1000;
    count = count +1;
    if count <= numdof
        query = 'n';
    end
end
Pknod = forceVec(nodeNum,1);
M =alpha*[13/35*1, 11/210*1^2, 9/70*1, -13/420*1^2; 11/210*1^2,
1/105*1^3, 13/420*1^2,-1/140*1^3; 9/70*1, 13/420*1^2, 13/35*1, -
11/210*1^2; -13/420*1^2, -1/140*1^3, -11/210*1^2, 1/105*1^3];

```

```

K = modulus*moment*[12/l^3, 6/l^2, -12/l^3, 6/l^2; 6/l^2, 4/l, -6/l^2,
2/l; -12/l^3, -6/l^2, 12/l^3, -6/l^2; 6/l^2, 2/l, -6/l^2, 4/l];

% This builds the global stiffness and mass matrices for the baseline
response.
mGlobal = zeros(numdof);
kGlobal = zeros(numdof);
s = 0;
for j = 1:numdof/2-1
    for ii=1:4
        for k=1:4
            kGlobal(ii+s, k+s) = K(ii, k) + kGlobal(ii+s, k+s);
            mGlobal(ii+s, k+s) = M(ii, k) + mGlobal(ii+s, k+s);
        end
    end
    s=s+2;
end

% This builds the global stiffness matrix for the modified response.
mGlobal2 = zeros(numdof);
kGlobal2 = zeros(numdof);
s = 0;
for j = 1:numdof/2-1
    for ii=1:4
        for k=1:4
            if j == 3 % Modified element number.
                kGlobal2(ii+s, k+s) = 1.5*K(ii, k) + kGlobal2(ii+s,
k+s);
            else
                kGlobal2(ii+s, k+s) = K(ii, k) + kGlobal2(ii+s, k+s);
            end
        end
    end
    s=s+2;
end

% This is to impose boundary conditions on the global mass and
stiffness
% matrices by restraining the applicable mode.
restrain = 'y';
while restrain == 'y'
    rNode = 1;
    dof1 = 2*rNode -1;
    dof2 = 2*rNode;
    kGlobal(dof1:dof2,:)=[];
    kGlobal(:,dof1:dof2)=[];
    kGlobal2(dof1:dof2,:)=[];
    kGlobal2(:,dof1:dof2)=[];
    mGlobal(dof1:dof2,:)=[];
    mGlobal(:,dof1:dof2)=[];
    forceVec(dof1:dof2)=[];
    restrain = 'n';
end

```

```

% This is the calculation for the tip displacement and tip rotation
using modal displacement.
[phi,lam] = eig(kGlobal,mGlobal);
[phi2,lam2] = eig(kGlobal2,mGlobal);
[row,col] = size(phi);

%Tip Displacement for a step load.
% Baseline response
c = 1;
[row1,col1]= size(t);
x = zeros(row,col1);
for jj = 1:row
for ii = t
qinit(jj,c) = exp(-dampingratio*sqrt(lam(jj,jj))*ii)*((-
dampingratio*phi(nodeNum-
2,jj)*Pknot/(sqrt(lam(jj,jj))^2*rootdamp))*sin(sqrt(lam(jj,jj))*rootdam
p*ii)...
-(phi(nodeNum-
2,jj)*Pknot/sqrt(lam(jj,jj))^2*cos(sqrt(lam(jj,jj))*rootdamp*ii))+phi(
nodeNum-2,jj)*Pknot/sqrt(lam(jj,jj))^2;
c = c+1;
end
c=1;
end
% Modified Response
count = 1;
[row2,col2]= size(t);
x2 = zeros(row,col2);
for jj = 1:row
for ii = t
qinit2(jj,count) = exp(-dampingratio*sqrt(lam2(jj,jj))*ii)*((-
dampingratio*phi2(nodeNum-
2,jj)*Pknot/(sqrt(lam2(jj,jj))^2*rootdamp))*sin(sqrt(lam2(jj,jj))*rootd
amp*ii)...
-(phi2(nodeNum-
2,jj)*Pknot/sqrt(lam2(jj,jj))^2*cos(sqrt(lam2(jj,jj))*rootdamp*ii))+ph
i2(nodeNum-2,jj)*Pknot/sqrt(lam2(jj,jj))^2;
count = count+1;
end
count=1;
end

%% Creating the vectors to be plotted
xplot = 0;
xplot2 = 0;
for ii = 1:row
xplot = xplot + qinit(ii,:)*phi(3 ,ii);
xplot2 = xplot2 + qinit2(ii,:)*phi2(3 ,ii);
end
% Plot these values against t to get a plot of the system response.

```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX H. MATLAB CODE FOR A NON-ADAPTIVE, NON- DERIVATIVE APPROXIMATION OF THE ALUMINUM CANTILEVERED BEAM (SUPPORTING FUNCTIONS INCLUDED)

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%This is the coarse solution to the MDOF beam problem that exhibited%
%conditional stability for varying stiffness modifications and time %
%steps.                                                                %
%Created by Keenan Coleman 8/25/14                                    %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Initializing the time variables:
t_start = 0;
t_end = .5;
delt = 0.01;
delt1 = 0.00001;
min_step = 0.00001;
time1 = t_start:delt1:t_end;
t_min = t_start:min_step:t_end;
nStep = length(t_min);
nStep2 = length(time1);
max_error = 0.0001;

% Initializing the initial conditions:
x_initial = zeros(4,1);
h_step = [0 (delt - t_start)];
x_coarse = [x_initial];
x_half = [x_initial];
x_fine = [x_initial];
x_approx = [x_initial];
base_result_plot = [x_initial];
count = 1;

t_step = [0 delt];
t_diff = [delt 0];
t_stepcount = t_start + delt;
count = 1;

% Initializing the physical parameters:
wn = sqrt(diag(lam));
fn = wn/2/pi;
wd = wn .* sqrt(1 - dampingratio.^2);
z = dampingratio*ones(row,1);
dk = 0.5*K;

% Initializing the forcing function:
f(row,:) = zeros(1,nStep);
f((row - 1),:) = 1000*ones(1,nStep);

% Calculate IRF matrix for baseline system (no mods):
H = fHmatrix_rev3(wn,wd,phi,z,t_min);

```

```

% Calculate baseline response using straight convolution with H matrix:
xBase = MDOF_base_rev3(H,f,min_step,nStep);

% Initiating the coarse solution algorithm:
while t_stepcount <= t_end

% Calling the necessary values of the baseline solution:
[base_result,base_result_half,base_result_plot] =
MDOF_base_call_rev3(t_min,
t_step(count),t_step(count+1),xBase,base_result_plot,count);

% Evaluating the kernel matrix over the adapted time-step:
[h_matrix] = h_kernel_rev3(wn,wd,phi,z,t_diff);

% Implementing the coarse solution:
[ x_coarse,x_coarsevalue ] = MDOF_coarse_rev3(h_matrix,
base_result,x_coarse,dk,h_step,count);

% Implementing a non-adaptive time-step...
count = count + 1
h_step(count+1) = delt;
t_step(count+1) = t_step(count) + delt;
t_stepcount = t_step(count+1);
t_stepvalue = t_step(count) + delt;
[r1,c1] = size(t_step);
for ii = 1:c1
t_diff(ii) = t_step(end) - t_step(ii);
end

end

% Change default axes fonts.
set(0,'DefaultAxesFontName','Arial')
set(0,'DefaultAxesFontSize',14)
% Change default text fonts.
set(0,'DefaultTextFontname','Arial')
set(0,'DefaultTextFontSize',14)
figure
plot(t_step(1:end-1),x_coarse(1,:), 'r-.',t,xplot,'k--',t,xplot2,'b')
legend('x*(t)','Unmodified (exact)','Modified (exact)')
xlabel('Time (sec)')
ylabel('x*(t) (modified displacement)')
title('x*(t) vs. Time')
grid on

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%This function evaluates the IRF matrix multiplied by phi and phi%
%transpose
%Created by Keenan Coleman on 8/25/2014
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [ H ] = fHmatrix_rev3(wn,wd,phi,z,t_min)

```

```

[r1,c1] = size(phi);
ModalHmatrix = zeros(r1,r1,length(t_min));
H = zeros(r1,r1,length(t_min));
for ii = 1:r1
    ModalHmatrix(ii,ii,:) = exp(-z(ii)*wn(ii)*t_min) .* sin(wd(ii) *
    t_min) / wd(ii);
end
for iStep = 1 : length(t_min)
    H(:, :, iStep) = phi * ModalHmatrix(:, :, iStep) * phi';
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% This function evaluates the kernel over the time difference vector%
%Created by Keenan Coleman on 8/25/2014 %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [h_matrix] = h_kernel_rev3(wn,wd,phi,z,t_diff)
phil = phi(3:6,:);
[r1,c2] = size (phil);
ModalHmatrix = zeros(c2,c2,length(t_diff));
h_matrix = zeros(r1,r1,length(t_diff));
% The kernel evaluated at the actual time-steps:
for ii = 1:c2
    ModalHmatrix(ii,ii,:) = exp(-z(ii)*wn(ii)*t_diff) .* sin(wd(ii) *
    t_diff) / wd(ii);
end
for iStep = 1 : length(t_diff)
    h_matrix(:, :, iStep) = phil * ModalHmatrix(:, :, iStep) * phil';
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%This function calls the baseline response value at the current%
%time-step. This value has been calculated outside of the loop.%
%Created by Keenan Coleman on 8/25/2014. %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [base_result,base_result_half,base_result_plot] =
MDOF_base_rev3(t_min,
t_step1,t_step2,xBase,base_result_plot,count)
c1 = find(t_min >= t_step2);
c2 = find(t_min >= (t_step1 + t_step2)/2);
c1 = min(c1);
c2 = min(c2);
base_result = xBase(3:6,c1);
base_result_plot(:,count+1) = base_result;
base_result_half = xBase(:,c2);
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%This function calculates the baseline response using the convolution.%
%Values will be called as necessary in the loop. %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [ xBase ] = MDOF_base_rev3(H,f,min_step,nStep)

```

```

[r2,c2] = size(f);
xBase = zeros(r2,nStep);
for i = 1 : r2
    for j = 1 : r2
        tmp = conv(squeeze(H(i,j,:)),f(j,:));
        xBase(i,:) = xBase(i,:) + tmp(1:nStep) * min_step;
    end
end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%This is the MDOF coarse solution. This is set up to go adaptive.%
%Created by Keenan Coleman on 8/25/2014.                                %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [ x_coarse,x_coarsevalue ] = MDOF_coarse_rev3(h_matrix,
base_result,x_coarse,dk,h_step,count)
[r1,c1] = size(h_matrix(:, :, 1));
x_coarse_sum = zeros(r1,1);
for ii = 1:count
    x_coarse_sum = x_coarse_sum + .5*(h_step(ii) +
h_step(ii+1))*h_matrix(:, :, ii)*dk*x_coarse(:,ii);
end
x_coarsevalue = base_result - x_coarse_sum;
x_coarse(:,count+1) = x_coarsevalue;
end

```

APPENDIX I. MATLAB CODE FOR A NON-ADAPTIVE, DERIVATIVE APPROXIMATION OF THE ALUMINUM CANTILEVERED BEAM (SUPPORTING FUNCTIONS INCLUDED)

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%This is the non-adaptive derivative approximation of the aluminum%
%cantilevered beam.                                           %
%Created by Keenan Coleman on 8/25/2014                      %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Initializing the time variables:
t_start = 0;
t_end = .1;
delt = 0.00001;
min_step = 0.00001;
t_min = t_start:min_step:t_end;
nStep = length(t_min);
max_error = 0.0001;

% Initializing the initial conditions:
x_initial = zeros(4,1);
h_step = [0 (delt - t_start)];
xdot_coarse = [x_initial];
x_vec = [x_initial];
xapprox_sum = [x_initial];
base_result_plot = [x_initial];
trap_base = [x_initial];
count = 1;

t_step = [0 delt];
t_diff = [delt 0];
t_stepcount = t_start + delt;
count = 1;

% Initializing the physical parameters:
wn = sqrt(diag(lam));
fn = wn/2/pi;
wd = wn .* sqrt(1 - dampingratio.^2);
z = dampingratio*ones(row,1);
delk = K;
dk = delk;

% Initializing the step-forcing function:
f(10,:) = zeros(1,nStep);
f(9,:) = 1000*ones(1,nStep);
f1 = zeros(row,1);
f1(9,1) = 1000;

tic
t_diff2 = fliplr(t_min);
[IRF] = MDOF_trap_fIRF(wn, z, phi, t_diff2);

```

```

[hdot_matrix] = vec_to_matrix(IRF,t_diff2, phi);
[index1, index2] = size(t_diff2);

% Initiating the adaptive algorithms from Dr. Neta's paper:
while t_stepcount <= t_end
% Evaluating the exact baseline value at the current step:
[trap_base,trap_base_value] =
base_exact(wn,wd,phi,z,f1,trap_base,count,t_stepcount);
% Evaluating the kernel matrix over the adapted time-step:
hdot_matrix1 = hdot_matrix(:, :, index2-count:end);
% Creating the x vector and approximation values required for the
% coarse solution
if count >=2
[x_vec, xapprox_vec,xapprox_sum] = MDOF_x_vector(count, x_initial,
h_step, x_vec, xdot_coarse,xapprox_sum);
end
% Implementing the coarse solution:
[ xdot_coarse,xdot_coarsevalue ] = MDOF_coarse_dot(hdot_matrix1
,trap_base_value,xdot_coarse,x_vec,dk,h_step,count,x_initial,xapprox_sum);
% Implementing a non-adaptive time-step...
count = count + 1
h_step(count+1) = delt;
t_round = round(10e5*(t_step(count) + delt))/10e5;
t_step(count+1) = t_round;
t_stepcount = t_step(count+1);
t_stepvalue = t_step(count) + delt;

end
toc
% Plotting the approximated solution to the exact solution and the
baseline
% solution:
figure
% Change default axes fonts.
set(0,'DefaultAxesFontName', 'Arial')
set(0,'DefaultAxesFontSize', 14)
% Change default text fonts.
set(0,'DefaultTextFontname', 'Arial')
set(0,'DefaultTextFontSize', 14)

set(gcf,'WindowStyle','Docked')
plot(t,xplot2,'k',t_step(1:end-2),x_vec(1,:), 'r--')
title('Non-adaptive MDOF (Derivative Approach) (x*(t))')
legend('Modified (Exact)', 'Approximation')
xlabel('Time (sec)')
ylabel('x*(t)')
grid on

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%This function calculates the exact baseline response at the%
%current time-step. %
%Created by Keenan Coleman on 8/25/2014 %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

```

```

function [trap_base,trap_base_value] =
base_exact(wn,wd,phi,z,f1,trap_base,count,t_stepcount)
[r1,c2] = size (phi);
ModalHmatrix1 = zeros(r1,r1);
% The baseline evaluated at the actual time-steps:
for ii = 1:r1
ModalHmatrix1(ii,ii) = exp(-z(ii)*wn(ii)*t_stepcount) .* sin(wd(ii) *
t_stepcount) / wd(ii);
end
hdot_matrix11 = phi * ModalHmatrix1 * phi'*f1;
hdot_matrix11(1:2) = [];
hdot_matrix11(5:8) = [];
trap_base_value = hdot_matrix11;
trap_base(:,count+1) = trap_base_value;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%This function calculates the IRF for a given mode over a      %
%time interval. The structure of this function was           %
%provided by Dr. Joshua Gordis and modified by Keenan Coleman%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [ModeIRF] = fModalIRF_MDOF_trap_F(wn, zeta, t)
    wd = wn * sqrt(1 - zeta^2);
    ModeIRF = (wd*cos(t*wd) - wn*zeta*sin(t*wd))./(exp(t*wn*zeta)*wd);

% Used with permission
function [IRF] = MDOF_trap_fIRF(Wn, Zeta, phi, time);
%
% Usage: [IRF] = fIRF(Wn, Zeta, phi, time);
%
% Creates matrix whose ROWS are the IRF constructed from
% modal parameters passed into the function.
%
% Function uses all modes passed in, and will generate all IRF for
unique
% (symmetric) input-output pairs for all rows in [phi].
% IRF are stored in "symmetric column storage" (See
fSymmetricStore.m)
%
% Wn:      Vector of natural frequencies (rad/sec)
%          If Wn(i) < 0.1, rigid body mode is assumed.
%
% Zeta:    Scalar, or vector damping ratio (scalar applied to all
modes)
%
% phi:     Mass normalized modal matrix.    Num rows = number of
coordinates
%
%                                     Num cols = number of modes
to be used.
%
% time:    Time. Row vector of sampling points for IRF evaluation.
%
% Written by Joshua H. Gordis, Ph.D.
% Rev #2 - zeta can now be a vector 1/18/06
%
```

```

%


---




---



ndof          = size(phi,1);
nmodes        = size(phi,2);
nsymcol       = ndof * (ndof + 1) / 2;           % Number of
columns
IRF           = zeros(nsymcol,length(time));     % Initialize
matrix
modeIRF       = zeros(1,length(time));
isymcols      = 0;                             % Will end up
being nsymcol

if length(Zeta) == 1;
    Zeta = Zeta * ones(nmodes,1);
end

for irows = 1 : ndof;
    for icols = irows : ndof;
        isymcols = isymcols + 1;
        for imodes = 1 : nmodes;

            modeIRF = fModalIRF_MDOF_trap_F(Wn(imodes), Zeta(imodes),
time);

            IRF(isymcols,:) = IRF(isymcols,:) +...
                phi(irows,imodes) * phi(icols,imodes) * modeIRF;

        end;                                     % End
    end;                                         % End
end;                                           % End
end;                                           % End

% End function fIRF.m

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%This function creates and builds the array of x values needed%
%to evaluate the coarse solution                               %
%Created by Keenan Coleman on 8/25/2014                        %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [x_vec, xapprox_vec,xapprox_sum] = MDOF_x_vector(count,
x_initial, h_step, x_vec, xdot_coarse,xapprox_sum)
[r1,c1] = size(xdot_coarse);
xelem_sum = zeros(r1,1) ;
% This creates the vector of x values needed in the coarse solution.
This
% is also the solution to the problem.
if count ==2
    xelem_sum = 0.5*(xdot_coarse(:,count-1) +
xdot_coarse(:,count))*h_step(count) + x_initial;

```

```

else
    xelem_sum = x_vec(:,count -1) + 0.5*(xdot_coarse(:,count-1) +
xdot_coarse(:,count))*h_step(count);
end
x_vec(:,count) = xelem_sum;
% This creates a vector of approximations needed in the coarse
solution.
xapprox_vec = 0.5*(xdot_coarse(:,count-1) +
xdot_coarse(:,count))*h_step(count);
xapprox_sum = xapprox_sum + xapprox_vec;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%This function evaluates the coarse approximation for the derivative%
%approach.                                                                %
%Created by Keenan Coleman on 8/25/2014                                  %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [ xdot_coarse,xdot_coarsevalue ] =
MDOF_coarse_dot(hdot_matrix,
base_result,xdot_coarse,x_vec,dk,h_step,count,x_initial,xapprox_sum)
[r1,c1] = size(hdot_matrix(:, :,1));
xdot_coarse_sum1 = zeros(r1,1);
lhs_coeff = eye(r1) + 0.25*h_step(end)^2*hdot_matrix(:, :,end)*dk;

for ii = 1:count
    xdot_coarse_sum1 = xdot_coarse_sum1 + .5*(h_step(ii) +
h_step(ii+1))*hdot_matrix(:, :,ii)*dk*x_vec(:,ii);
end

xdot_coarse_sum3 = 0.5*h_step(end)*hdot_matrix(:, :,end)*dk*xapprox_sum;
xdot_coarse_sum4 =
0.25*h_step(end)^2*hdot_matrix(:, :,end)*dk*xdot_coarse(:,end) -
0.5*h_step(end)*hdot_matrix(:, :,end)*dk*x_initial;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%This function takes the vector of IRF and creates a%
%lower triangular matrix.                                                %
%Created by Keenan Coleman on 8/25/2014                                  %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [hdot_matrix] = vec_to_matrix(IRF,t_diff, phi)
[r1,c1] = size(t_diff);
[r2,c2] = size(phi);
hdot_matrix = zeros(r2,c2,c1);
for ii = 1:c1
    A = zeros(r2);
    ind = find(tril(ones(10)));
    A(ind) = IRF(:,ii);
    A = A + tril(A,-1)';
    hdot_matrix(:, :,ii) = A;
end
hdot_matrix(1:2, :, ) = [];
hdot_matrix(5:8, :, ) = [];
hdot_matrix(:, 1:2, ) = [];
hdot_matrix(:, 5:8, ) = [];

```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX J. MATLAB CODE FOR THE MATRIX FORMULATION TO THE DERIVATIVE OF THE INTEGRAL FORMULATION FOR TRANSIENT STRUCTURAL ANALYSIS (SUPPORTING FUNCTIONS INCLUDED)

Many of the functions used for this program can be found in Appendix I.

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%This is the matrix formulation for the derivative of the integral%
%formulation for transient structural synthesis.                      %
%Created by Keenan Coleman on 8/25/2014                             %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Initializing the time variables:
delt = 0.00001;
t_start = delt;
t_end = .1-delt;
t_diff = t_start:delt:t_end;
t_diff2 = 0:delt:t_end;
nStep = length(t_diff);
nStep2 = length(t_diff2);
% Initializing the physical parameters:
wn = sqrt(diag(lam));
fn = wn/2/pi;
wd = wn .* sqrt(1 - dampingratio.^2);
z = dampingratio*ones(row,1);
delk = K;
dk = delk;

% Initializing the step-forcing function:
f(10,:) = zeros(1,nStep);
f(9,:) = 1000*ones(1,nStep);
f1 = zeros(row,1);
f1(9,1) = 1000;

% Calculating the array of matrices to be called in the matrix
formulation
[IRF] = MDOF_trap_fIRF(wn, z, phi, t_diff2);
[hdot_matrix] = vec_to_matrix(IRF,t_diff2, phi);
[ hdot_dk ] = hdotmultiplydk( hdot_matrix,dk,nStep2 );
hdot_dk_zero = eye(4) + hdot_dk(:, :, 1)*.25*delt^2;
hdot_dk = delt^2*hdot_dk;
hdot_dk(:, :, 1) = 0.5*hdot_dk(:, :, 1);

% Building the matrix
nCset= 4;
A2 = zeros(nCset*nStep2,nCset*nStep2);
elementnumber = [hdot_dk_zero];
colcount = 1:4;
columnbuild = 0;
tic

```

```

for intstep5 = 2:nStep2
    elementsum = zeros(4,4);
    colcount = colcount + 4;
    for intstep6 = 1:intstep5
        if intstep6 == intstep5
            elementsum = elementsum + 0.5 * hdot_dk(:, :, intstep6);
        else
            elementsum = elementsum + hdot_dk(:, :, intstep6);
        end
    end
    elementnumber(:, colcount) = elementsum;
    columnbuild = columnbuild + 1
end
% toc
rows2 = -3:0;
cols2 = -3:0;
RowCnt2 = 0;
ColCnt2 = 0;
matrixbuild = 0;
% tic
for iStep2 = 1 : nStep2;
    RowCnt2 = RowCnt2 + 1;
    rows2 = rows2 + 4;
    cols2 = -3 : 0;
    cols3 = rows2;

    for jStep2 = 1 : iStep2;
        ColCnt2 = ColCnt2 + 1;
        cols2 = cols2 + 4;
        [RowCnt2 ; ColCnt2];
        A2(rows2, cols2) = elementnumber(:, cols3);
        cols3 = cols3 - 4;

    end
    matrixbuild = matrixbuild + 1
end
% toc

% Building the vector:
cnt2 = 0;
rowindex = [1:4];
x_initial = zeros(4,1);
trap_base = [x_initial];
% tic
for intStep2 = t_start:delt:.1
    [trap_base, trap_base_value] =
base_exact(wn, wd, phi, z, fl, trap_base, count, intStep2);
    base_vec(rowindex, 1) = trap_base_value;
    rowindex = rowindex + 4;
end
% toc
bb = 0;
% tic
velocity = A2\base_vec;
toc

```

```

% Creating and plotting the velocity vector:
velocity2 = [0];
for intstep3 = 1:nStep
    velocity2(1,intstep3+1) = velocity(4*intstep3-1,1);
end
plot(t_diff2, velocity2)

% Plotting the displacement vector:
displacement = [0];
for intstep4 = 2:nStep2
    displacement2 = 0.5*delt*(velocity2(intstep4-1) +
velocity2(intstep4));
    displacement(1,intstep4) = displacement(1,intstep4-1) +
displacement2;
end

% Change default axes fonts.
set(0,'DefaultAxesFontName', 'Arial')
set(0,'DefaultAxesFontSize', 14)
% Change default text fonts.
set(0,'DefaultTextFontname', 'Arial')
set(0,'DefaultTextFontSize', 14)
plot(t,xplot2,'-*k',t_diff2,displacement,'--r')
title('Matrix Formulation of the Derivative of the Synthesis
Equation');
xlabel('Time (seconds)');
ylabel('x*(t)')
legend('Jerri','exact')
toc

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%This function multiplies the array of h_dot matrices by the elemental%
%stiffness matrix                                                    %
%Created by Keenan Coleman on 8/25/2014                                %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [ hdot_dk ] = hdotmultiplydk( hdot_matrix,dk,nStep )

for ii = 1:nStep
    hdot_dk(:, :, ii) = hdot_matrix(:, :, ii)*dk;
end

```

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

- [1] A.T. Lonseth, "Sources and applications of integral equations," *SIAM Rev.*, vol. 19, no. 2, pp. 241–278, Apr. 1977.
- [2] J.H. Gordis, "Integral equation formulation for transient structural synthesis," *AIAA J.*, vol. 33, no. 2, pp. 320–324, Feb. 1995.
- [3] J.H. Gordis, and B. Neta, "An adaptive method for the numerical solution of Volterra integral equations," in *World Sci. and Eng. Soc. Intl. Conf.*, Athens, Greece, 2000, pp. 1–8.
- [4] P. Linz, *Analytic and Numerical Methods for Volterra Equations*, Philadelphia, PA: SIAM, 1985, pp. 95–128.
- [5] H. Brunner, and P.J. van der Houwen, *The Numerical Solution of Volterra Equations*, Amsterdam, Netherlands: Elsevier Sci., 1986, pp. 409–499.
- [6] C.T.H. Baker, *The Numerical Treatment of Integral Equations*, Oxford, United Kingdom: Oxford Univ. Press, 1977, pp. 788–825.
- [7] P. Linz, *Analytic and Numerical Methods for Volterra Equations*, Philadelphia, PA: SIAM, 1985, pp. 45–46.
- [8] A.J. Jerri, *Introduction to Integral Equations with Applications*, 2nd ed., New York: Wiley-Interscience, 1999, p. 24–30.
- [9] R.R. Craig and A.J. Kurdila, *Fundamentals of Structural Dynamics*, 2nd ed., Hoboken, NJ: Wiley, 2006, pp. 125.
- [10] Y.W. Kwon and H. Bang, "Beam and frame structure," in *The Finite Element Method Using MATLAB*, F. Krieth, Ed., 2nd ed. New York: CRC Press, 2000, pp. 237–310.
- [11] A.J. Jerri, *Introduction to Integral Equations with Applications*, 2nd ed., New York: Wiley-Interscience, 1999, pp. 133–164.

THIS PAGE INTENTIONALLY LEFT BLANK

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
Ft. Belvoir, Virginia
2. Dudley Knox Library
Naval Postgraduate School
Monterey, California